



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BarcelonaTech

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Departamento de Lenguajes y Sistemas Informáticos (LSI)

Trabajo Final de Grado
Grado en Ingeniería Informática
Especialidad en Computación

Red social como interfaz de servicios asistenciales para personas mayores

Miguel Pérez Pasalodos

miguel.perez.pasalodos@est.fib.upc.edu

Director:

Cristian BARRUÉ SUBIRANA

Ponente:

Ulises CORTÉS

Defensa: 4 de febrero de 2014

RESUMEN

Los avances en las diferentes ramas de la medicina y la tecnología han hecho incrementar la esperanza de vida, que no ha parado de aumentar desde hace un tiempo. Esto provoca una deformación en la pirámide poblacional, haciendo que exista una mayor proporción de personas de avanzada edad. Estas personas son más propensas a sufrir ciertas enfermedades, que a su vez pueden generar discapacidades tanto físicas como cognitivas.

Muchos de estos ancianos con discapacidades utilizan dispositivos que hacen sus vidas más fáciles. No obstante, la mayoría de estos dispositivos son muy simples. La tecnología, no obstante, brinda cada día una mayor posibilidad de llevar procesadores a más niveles de la sociedad. Es posible, por tanto, aumentar las capacidades de estos dispositivos añadiendo sensores y un computador sencillo para procesar los datos.

Las personas relacionadas con los pacientes también se ven afectadas por estas dolencias. El mayor problema es la falta de información. Por ejemplo, un médico puede observar a su paciente durante una consulta, pero no en su vida cotidiana. Los nuevos dispositivos pueden ofrecer esta información usando sensores, pero estos datos suelen ser difíciles de analizar.

Este proyecto ofrece una herramienta mediante la cual se pueden procesar estos datos y, finalmente, mostrarlos de diferentes maneras en una red social médica. El sistema permite añadir nuevos dispositivos (servicios asistenciales) que, haciendo uso de un sistema basado en agentes, procesarán los datos. Este decide de forma proactiva qué información hay que mostrar en la red, a quién y de qué forma mostrarla. La información mostrada puede variar desde notificaciones de peligro hasta informes de actividad.

Haciendo uso de este sistema, se han desarrollado servicios de ejemplo para un caminador inteligente, de nombre i-Walker, que ha sido desarrollado en la UPC. El sistema multi-agente se encarga de procesar los datos ofrecidos por los diferentes sensores (velocidad, fuerzas, inclinación, etc.) y genera la información que crea necesaria. Esta información puede ser enviada tanto a la red social como a los agentes de los usuarios relacionados con el paciente.

Palabras clave: sistemas multi-agente, redes sociales, servicios asistenciales, i-Walker, BDI, Python, PHP.

ABSTRACT

Progress in some fields of medicine and technology have increased the life expectancy, which has been growing for some time. This is causing a deformation in the population pyramid, resulting in a greater proportion of elderly people. Those elders are more prone to some maladies, which can cause both cognitive and physical disabilities.

Many of these elders suffering disabilities use assistive devices to make their lives easier. Nonetheless, most of them are very simple. New technologies, however, can bring computers to a new range of devices. Therefore, it is possible to enhance the capabilities of those assistive devices. For example, this has been shown in adding sensors to collect data about the patient and a simple computer to process them.

People related to patients are also affected by their disabilities. One of the biggest problems is the lack of information. In fact, a doctor can observe his patients during a consultation, but not in their everyday life. New devices can provide this information using sensors, but the data provided is usually hard to analyse.

This project provides a tool to process this data and, eventually, to display it in different ways on a medical social network. Developers will be able to add new devices (assistive services), which will process the data using an agent based system. It can proactively decide which information is going to be shown on the network, to whom the information will be displayed and how to show it. The shown information include warning notices to activity reports.

Using the previous systems, some sample services have been developed using the i-Walker device, a smart walking frame developed at the UPC. The multi-agent system processes the data provided by the walker sensors (speed, forces, inclination, etc.) and generates the necessary information, which can be sent to the users that are related to the patient on the social network.

Keywords: multi-agent systems, social networks, assistive aids, i-Walker, BDI, Python, PHP.

RESUM

Els avenços en les diferents branques de la medicina i la tecnologia han fet incrementar l'esperança de vida, que no ha parat d'augmentar des de fa un temps. Això provoca una deformació en la piràmide poblacional, fent que hi hagi una major proporció de persones d'edat avançada. Aquestes persones són més propenses a patir certes malalties, que poden generar discapacitats tant físiques com cognitives.

Molts d'aquests ancians amb discapacitats utilitzen dispositius que fan les seves vides més fàcils. No obstant això, la majoria d'aquests dispositius són molt simples. La tecnologia, però, ofereix cada dia una major possibilitat de portar processadors a més nivells de la societat. És possible, per tant, augmentar les capacitats d'aquests dispositius afegint sensors que agafin diferents dades i un computador senzill que les processi.

Les persones relacionades amb els pacients també es veuen afectades per aquestes malalties. El problema més gran és la falta d'informació. Per exemple, un metge pot observar al seu pacient durant una consulta, però no en la seva vida diària. Els nous dispositius poden oferir aquesta informació fent ús de sensors, però normalment aquestes dades solen ser difícils d'analitzar.

Aquest projecte ofereix una eina mitjançant la qual es poden processar aquestes dades i, finalment, mostrar-les de diferents maneres en una xarxa social mèdica. El sistema permet afegir nous dispositius (serveis assistencials) que, fent ús d'un sistema basat en agents, processaran les dades. Aquest decideix de forma proactiva quina informació cal mostrar a la xarxa, a qui i de quina forma mostrar-la. La informació mostrada pot variar des de notificacions de perill fins a informes d'activitat.

Fent ús d'aquest sistema, s'han desenvolupat serveis d'exemple per a un caminador intel·ligent, de nom i-Walker, que ha estat desenvolupat a la UPC. El sistema multi-agent s'encarrega de processar les dades dels diferents sensors (velocitat, forces, inclinació, etc.) i genera la informació que cregui necessària. Aquesta informació es pot enviar a la xarxa social o als agents dels usuaris relacionats amb el pacient.

Paraules clau: sistemes multi-agent, xarxes socials, serveis assistencials, i-Walker, BDI, Python, PHP.

Índice general

1. Introducción	1
1.1. Motivación y justificación del proyecto	1
1.2. Objetivos	2
1.3. Límites y exclusiones	3
1.4. Escenario	4
1.5. Contribuciones de este proyecto	5
1.6. Estructura del documento	5
2. Estado del Arte	7
2.1. Redes sociales	7
2.2. Sistemas multi-agente	8
2.3. Integración de redes sociales y el sistema multi-agente	9
2.4. Servicios asistenciales	9
3. Red social	10
3.1. Vértices	11
3.1.1. Personas	12
3.1.2. Dispositivos asistenciales	13
3.1.3. Informes	13
3.1.4. Mensajes	14
3.2. Relaciones entre vértices	14
3.2.1. Relaciones entre usuarios	15
3.2.2. Relación entre pacientes y dispositivos	17
3.2.3. Otras relaciones	17
4. Sistema multi-agente	19
4.1. Modelo de software	21
4.1.1. BDI	21
4.2. Distribución de los agentes	22
4.2.1. Dispositivos asistenciales	22
4.2.2. Personas	22
4.2.3. Gestor de agentes	23
4.3. Arquitectura	25
4.3.1. Base	25
4.3.2. Agentes personales	26
4.3.3. Dispositivos asistenciales	29
4.3.4. Gestor de agentes	30

5. Dispositivo de ejemplo: i-Walker	35
5.1. Datos generados por el i-Walker	37
5.2. Pruebas con el caminador	38
5.2.1. Medición de ruido con el caminador parado	39
5.2.2. Línea recta con paradas	39
5.2.3. Curvas	41
5.2.4. Curvas con ayuda	42
5.2.5. Curvas con ejercicio	42
5.2.6. Frenos	44
5.2.7. Acelerómetros	45
5.2.8. Fuerzas	45
5.2.9. Prueba real	46
5.3. Procesado de datos	47
6. Implementación de la red social	49
6.1. Motor para la red social	49
6.2. Elgg	50
6.2.1. Sistema	51
6.2.2. Modelo de datos	51
6.2.3. Desarrollo de añadidos y modificaciones	55
6.3. Arquitectura de los <i>plugins</i>	57
6.4. Usuarios, roles y relaciones	57
6.4.1. Roles	58
6.4.2. Relaciones	59
6.4.3. Páginas	61
6.5. Informes de los pacientes	62
6.5.1. Páginas	63
6.6. Dispositivos asistenciales	63
6.6.1. Páginas	64
6.7. Otros <i>plugins</i>	65
6.7.1. rsActivity	65
6.7.2. rsProfile	66
6.7.3. rsMessages	66
6.7.4. rsManage	67
6.7.5. rsTheme	67
7. Implementación del sistema multi-agente	68
7.1. SPADE	68
7.1.1. Comportamientos	68
7.1.2. Mensajes	70

7.1.3. AMS y DF	71
7.1.4. Base de conocimiento	71
7.2. Jerarquía de ficheros	72
7.3. Agentes	73
7.3.1. Agent.py	73
7.3.2. BDIAgent.py	74
7.3.3. Person.py	77
7.3.4. Device.py	77
7.3.5. DeviceBehaviours.py	78
7.3.6. Manager.py	79
7.3.7. PatientAgent.py	81
7.3.8. Otros agentes personales	82
8. Implementación de la interfaz	84
8.1. Peticiones en la red social	86
8.2. Peticiones en el sistema multi-agente	92
8.3. Seguridad	94
9. Implementación de los servicios: i-Walker	96
9.1. Jerarquía de directorios	96
9.2. Agente	98
9.3. Comportamientos	100
9.3.1. Gestión de mensajes	101
9.3.2. Gestión de peligros y avisos	101
9.3.3. Sesiones	102
9.3.4. Composición de informes	102
10. Resultados	106
10.1. Conexión entre sistemas	106
10.2. Informes de sesión	111
10.3. Notificaciones de peligro	111
11. Conclusiones y trabajo futuro	114
11.1. Red social	114
11.2. Sistema multi-agente	116
11.3. Interfaz entre sistemas	117
11.4. Dispositivos asistenciales	117
11.5. Conclusiones finales	118
12. Gestión del proyecto	119
12.1. Planificación	119

12.1.1. Planificación original	119
12.1.2. Cambios en la planificación	124
12.2. Presupuesto	125
12.2.1. Presupuesto original	125
12.2.2. Cambios en el presupuesto	127
12.3. Metodología de desarrollo	129
12.3.1. Criterios de validación	130
12.3.2. Métodos de trabajo y organización	130
12.3.3. Cambios en las metodologías	131
12.4. Impacto	131
12.4.1. Impacto social	131
12.4.2. Impacto ambiental	132
12.4.3. Impacto económico	133
12.4.4. Viabilidad y sostenibilidad	133
12.5. Regulaciones	133
12.5.1. Bibliotecas y <i>frameworks</i>	134
12.5.2. Algoritmos y paradigmas	135
12.5.3. Uso de software	135
12.5.4. Protección de datos personales	136
A. Instalación	139
A.1. Red social	139
A.2. Sistema multi-agente	140
Glosario	145
Siglas	147

Índice de figuras

1.1.	Esquema de la arquitectura básica de los sistemas	2
3.1.	Ejemplo del grafo de una red social. Fuente: Griff's Graphs	11
3.2.	Ejemplo del diseño de los vértices de la red y sus relaciones	15
4.1.	Esquema de un agente en su entorno. El agente capta información del entorno y produce acciones que afectan a este. Fuente: [50]	19
4.2.	Arquitectura del sistema multi-agente	25
4.3.	Esquema de la arquitectura del gestor de agentes	33
5.1.	Vista frontal y posterior del caminador i-Walker	35
5.2.	Boxplots mostrando los valores de ruido de los sensores de fuerza del caminador: manillares (izquierdo y derecho) y fuerza normal	40
5.3.	Valores de los sensores de velocidad del caminador en línea recta con diversas paradas	41
5.4.	Media de fuerzas producidas por el caminador al andar zigzagueando	42
5.5.	Media de fuerzas producidas por el caminador al andar zigzagueando con ayuda (λ) máxima	43
5.6.	Media de fuerzas producidas por el caminador al andar zigzagueando con un valor de ejercicio (ν) máximo	43
5.7.	Resultados de los sensores en los frenos del caminador y velocidades durante una prueba de frenos	44
5.8.	Valores obtenidos de los acelerómetros del caminador en una prueba de inclinaciones	46
5.9.	Fuerzas (x , y , z) captadas por el i-Walker durante una prueba de fuerzas	47
6.1.	Modelo de datos de Elgg	52
6.2.	Arquitectura de las modificaciones realizadas Elgg para el proyecto .	58
7.1.	Esquema de la arquitectura de SPADE	69
8.1.	Esquema de la interfaz entre red social y sistema multi-agente	85
10.1.	Pasos para la creación de un paciente en la red social	107
10.2.	Interfaz para añadir relaciones a un usuario	108
10.3.	Creación y asignación de un dispositivo en la red social	109
10.4.	Proceso de desasignación de un dispositivo a un paciente en la red social	110
10.5.	Ejemplo de informe de un ejercicio generado por el i-Walker	112
10.6.	Ejemplo de informe de una sesión generado por el i-Walker	112

10.7. Ejemplo de notificación en la red social	113
12.1. Diagrama de Gantt de la planificación temporal original	122

Índice de tablas

5.1. Referencia de los datos registrados por el i-Walker	38
5.2. Referencia de los eventos procesados por el i-Walker	39
12.1. Coste de los recursos humanos en el proyecto	127
12.2. Estimación inicial de los costes totales del proyecto (incluyendo adquisiciones)	128
12.3. Estimación inicial de los costes totales imputables al proyecto	128
12.4. Costes finales totales del proyecto (incluyendo adquisiciones)	129
12.5. Costes finales totales imputables al proyecto	130

Índice de códigos

6.1 Ejemplos de operaciones con relaciones entre entidades (Elgg)	54
6.2 Ejemplo de operaciones con roles (Elgg)	59
6.3 Modelo para la gestión de relaciones en la red social	60
6.4 Modelo para la gestión de informes en la red social	62
6.5 Modelo para la gestión de dispositivos en la red social	64
7.1 Ejemplo de deducción de conocimiento en SPADE	72
8.1 Formato de salida de la función para obtener grupos de la red	87
8.2 Ejemplo del parámetro <code>data</code> para la función <code>users.reports.new</code> (generación de gráficos)	88
8.3 Funciones en la interfaz de la red social	90
8.4 Referencia de las funciones expuestas por el sistema multi-agente	93
9.1 Ejemplo del fichero de configuración de dispositivos	97
9.2 Ejemplo de <code>__init__.py</code> para el caminador i-Walker	97

1

Introducción

Las continuas mejoras referentes a la medicina están provocando un gran cambio demográfico. Estos avances están provocando un aumento de la longevidad, lo que resulta en un mayor porcentaje de personas mayores. Sin embargo, aunque la longevidad vaya en aumento, no significa necesariamente que la calidad de vida de estas personas mejore de la misma manera. El envejecimiento de la población, por tanto, conlleva un aumento de las enfermedades relacionadas con la edad, que a su vez generan discapacidades tanto físicas como cognitivas.

El mundo de la tecnología ha ido avanzando también a pasos agigantados. Estos avances permiten llevar computadores a elementos que hace unos años no podrían haber sido imaginados. Estas nuevas tecnologías puede aplicarse de muchas maneras. Una de ellas consiste en crear dispositivos inteligentes para ayudar a personas mayores con alguna discapacidad. La tecnología, por tanto, se usa para intentar mejorar un problema humano como es el envejecimiento.

1.1 Motivación y justificación del proyecto

Durante el transcurso de mis estudios he aprendido diferentes aspectos sobre la informática. Empecé estudiando algunos fundamentos matemáticos y el funcionamiento interno de un computador. A medida que pasaba el tiempo, aprendí a diseñar buenos sistemas, a analizar su rendimiento y a mejorarlo. Cada vez que aprendía algo nuevo, me preguntaba cómo podía aplicarlo para hacer más llevadera la vida de alguien.

Pocas asignaturas intentaban mostrar este aspecto de la informática. No obstante, en cierto momento empecé a interesarme por la Inteligencia Artificial. En esas asignaturas había muchos ejemplos sobre cómo aplicar técnicas para mejorar la sociedad en su conjunto y no sólo aplicado a un único individuo, algo por lo que tenía mucho interés. En particular, la ingeniería del conocimiento y los sistemas multi-agente me enseñaron que un sistema informático podía regular el consumo eléctrico de una ciudad, o incluso diagnosticar pacientes.

En ese momento tenía claro que mi trabajo de final de grado sería sobre IA, pero el campo ofrecía tantas posibilidades que no tenía claro qué hacer. Una búsqueda entre

los miembros del grupo KEML¹ me llevó a proyectos muy interesantes, pero que no se acercaban a lo que yo buscaba realmente.

Cuando estaba barajando la posibilidad de realizar alguna de estas propuestas, Cristian Barrué me comentó un proyecto que tenía en mente. Consistía en realizar un sistema multi-agente para gestionar dispositivos asistenciales, los cuales servirían para mejorar la vida de personas con alguna discapacidad. Todo ello iría representado en una red social, para que los usuarios pudieran acceder a ese procesado de datos de una forma más amigable. El proyecto aplicaba todo aquello que estaba buscando: sistemas multi-agente y tecnología aplicada a la sociedad.

1.2 Objetivos

El objetivo principal del proyecto es ofrecer un modelo para desarrollar un sistema multi-agente integrado en una red social médica. El sistema deberá ofrecer una herramienta para desarrollar dispositivos (que tomarán el nombre de servicios asistenciales) inteligentes que ayuden a los pacientes con alguna discapacidad. El sistema multi-agente estará conectado a una red social, ya que las discapacidades del paciente son un problema colectivo. Esto es así porque esta discapacidad afecta a todas aquellas personas que estén relacionadas con el afectado, desde sus familiares más cercanos al personal sanitario que lleve su tratamiento. La red deberá poder usarse de diferentes maneras dependiendo del tipo de usuario que la esté usando.

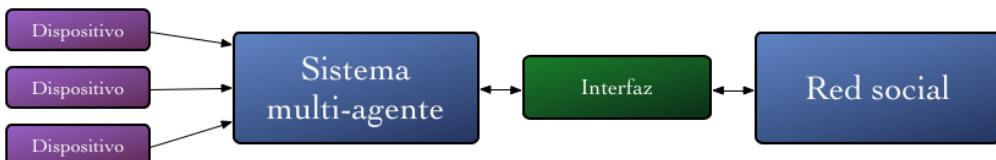


Figura 1.1: Esquema de la arquitectura básica de los sistemas

Los dispositivos con sensores ofrecen algunas dificultades. Uno de sus problemas es que los datos generados suelen ser simples números, muy poco amigables para las personas. Para solucionar este problema, el proyecto tiene el objetivo de desarrollar un sistema de IA que procese esos datos y delibere sobre la importancia de estos para los diferentes actores en la red social. Así mismo, también será necesario desarrollar una interfaz que conecte ambos sistemas, de forma que no se produzcan incongruencias entre ambos. La figura 1.1 muestra un esquema simple de esta arquitectura.

¹Grupo de ingeniería del conocimiento y aprendizaje automático

El proyecto tiene unos objetivos de desarrollo, pero los diferentes sistemas desarrollados tienen también otra serie de objetivos funcionales con sus usuarios. Estos objetivos son:

1. Ofrecer a los cuidadores herramientas para que puedan dar un tratamiento más concreto a cada uno de sus pacientes. Esto se logrará aportando información detallada sobre la condición médica de los pacientes. La información será generada de forma automática a través de los servicios asistenciales.
2. Alertar a los médicos si los sistemas detectan una urgencia médica.
3. Ayudar a mejorar la interacción entre los pacientes, cuidadores y familiares.

Otro de los objetivos del proyecto es aplicar esta herramienta a un dominio más específico. En concreto, en el proyecto se incluirán servicios asistenciales de ejemplo para un dispositivo, el caminador robótico i-Walker [1] desarrollado en la UPC. Estos servicios harán uso de los datos ofrecidos por los sensores del caminador para intentar mejorar el tratamiento del paciente, informando a todas las personas que se vean afectadas en mayor o menor medida por la discapacidad de este.

1.3 Límites y exclusiones

Debido al tiempo limitado disponible para finalizar el proyecto, es necesario establecer ciertos límites o exclusiones para el desarrollo. Un proyecto demasiado ambicioso puede acabar en fracaso. En el caso de este proyecto, aún manteniendo todos los objetivos, marca los siguientes límites:

- Algunas de las características más avanzados de las redes sociales quedan fuera del alcance del proyecto. Sin embargo, serían un aspecto muy a tener en cuenta en un posible trabajo futuro. Las características que se ha decidido excluir son: grupos, eventos, mensajería instantánea, juegos didácticos y publicación de material multimedia.
- Aunque la herramienta permitirá desarrollar cualquier tipo de servicio, en este proyecto sólo se incluirán aquellos relacionados con el caminador robótico i-Walker.
- Debido a la gran cantidad de tiempo necesario para desarrollar un motor de redes sociales completo, se preferirá la utilización y modificación de un motor de código libre ya existente. Además de ahorrar tiempo, el sistema podrá ofrecer una red social más madura y estable.

Así pues, el proyecto incluye tanto el diseño de un modelo de sistema multi-agente incrustado en una red social como la implementación de servicios basados en este modelo. De forma resumida, los objetivos del proyecto son:

1. Diseño e implementación de una **red social** con servicios de comunicación básicos entre sus usuarios. La red debe ofrecer las herramientas necesarias para la gestión de diferentes tipos de usuarios y dispositivos asistenciales.
2. Diseño e implementación de un **sistema multi-agente**. El sistema se encargará de captar y procesar los datos de diferentes dispositivos asistenciales y podrá, entre otras cosas, enviar notificaciones de forma proactiva a la red. Sus objetivos se centran en predecir y enviar alertas a la red, generar informes de seguimiento e incentivar positivamente a los usuarios fomentando la interacción entre ellos. Este sistema ofrecerá una base para añadir dispositivos, lo que permitirá a los desarrolladores modificar su comportamiento incluyendo nuevos módulos.
3. Diseño e implementación de una **interfaz** que permitira la interacción entre los dos sistemas citados anteriormente.
4. Haciendo uso del modelo anterior, **desarrollar servicios de ejemplo** para la red que se basarán en el caminador robótico i-Walker.

1.4 Escenario

Para entender mejor los objetivos del proyecto, esta sección describe un ejemplo completo del uso que deberían permitir los sistemas. El ejemplo explica una situación ficticia que podría darse de forma real, incluyendo un paciente con problemas para caminar, un familiar suyo y un médico que les atiende. El proceso sería el siguiente:

1. Pedro y su padre Juan acuden al médico. El objetivo de la visita es realizar una revisión médica a Juan.
2. Durante la revisión, el doctor Marc observa que Juan camina con dificultad y usando un bastón. Les comenta que están usando un nuevo sistema con dispositivos inteligentes que pueden mejorar el tratamiento y la vida de Juan. Les explica también que los dispositivos están conectados a una red social que recibe los datos de los dispositivos, informa sobre la actividad de estos y alerta sobre posibles peligros. A ambos les parece bien la idea, y Marc receta un caminador inteligente a Juan.

3. Marc da de alta a Juan y a Pedro en la red social, y crea la relación entre los nuevos usuarios (paciente y familiar) y su propio usuario. Además, asigna uno de los caminadores libres a Juan.
4. Juan utiliza el dispositivo por algún tiempo. Durante este tiempo, el dispositivo ha generado informes de la actividad de Juan, que Marc ha ido estudiando. En cierto momento, el dispositivo indica que Juan está realizando demasiada fuerza al caminar, por lo que decide informar a Marc y a Pedro.
5. Marc y Pedro reciben una alerta en la red social, que les informa sobre el exceso de fuerzas que ha realizado Juan usando el caminador.
6. Finalmente, Marc decide que Juan debería volver a la consulta para ver si existe algún nuevo problema de salud, por lo que envía un mensaje a través de la red a Pedro, el familiar responsable de Juan.

1.5 Contribuciones de este proyecto

El proyecto presenta un modelo para el desarrollo de redes sociales médicas basadas en dispositivos asistenciales inteligentes. Dicho modelo integra un sistema multi-agente en una red social, aspecto en el cual es pionero. Durante el proyecto, sólo se aplica a un dominio específico (caminador robótico i-Walker), aunque ha sido diseñado para poderse aplicar de diferentes maneras. Por tanto, a nivel técnico, el proyecto contribuye ofreciendo una herramienta para el desarrollo de agentes integrados en redes sociales médicas. De esta forma, otros desarrolladores pueden tanto ampliar las funcionalidades específicas de este proyecto como desarrollar nuevas redes sociales. En el aspecto social, el proyecto contribuye a mejorar la vida de todos sus posibles usuarios, haciendo que el tratamiento de los pacientes sea más eficaz y más fácil de realizar, además de mantener informados al resto de usuarios relacionados.

1.6 Estructura del documento

El resto de esta memoria está estructurado de la siguiente manera: el capítulo 2 muestra el estado del arte en los diferentes aspectos relacionados con el proyecto, incluyendo redes sociales, sistemas multi-agente y la conexión entre ambos. A continuación, el capítulo 3 estudia las características y el diseño de la red social. El capítulo 4 hace lo mismo para el sistema multi-agente, y el capítulo 5 se centra en el estudio de los datos ofrecidos por el caminador i-Walker (en los que se basarán los

servicios de ejemplo). Más adelante, los capítulos 6, 7, 8 y 9 se centran en detallar la implementación de lo concluído en los capítulos anteriores. Después, el capítulo 10 muestra algunos de los resultados obtenidos aplicando todos los sistemas. Más adelante, el capítulo 11 agrupa todas las conclusiones extraídas en los capítulos anteriores. Finalmente, el capítulo 12 detalla algunos aspectos importantes que se han llevado a cabo en la gestión del proyecto. El documento finaliza con un anexo donde se detalla cómo instalar e iniciar los sistemas.

2

Estado del Arte

Hay dos partes fundamentales dentro del desarrollo del proyecto: redes sociales y sistemas multi-agente. Existen gran cantidad de estudios e implementaciones para cada una de estas partes por separado, pero no sobre la integración de un sistema multi-agente dentro de una red social. Por tanto, ni la creación de la red social ni el sistema multi-agente requerirá un trabajo de investigación, pero sí la integración entre ambos. En las siguientes secciones se estudia más detalladamente la situación para cada una de las partes.

2.1 Redes sociales

Una red social [7] es un sistema informático que permite a sus usuarios tener algún tipo de relación entre ellos (amigos, familiares, etc.), comunicarse (envío de mensajes, por ejemplo) y compartir información de forma pública o semi-pública (sólo con los usuarios relacionados). Este diseño permite representar una estructura social de cualquier tipo, por lo que las redes sociales pueden usarse para diferentes entornos. Por ejemplo, se puede desarrollar una red social sólo para los miembros de una empresa o, como en el caso de este proyecto, una red social médica. En este último caso, los usuarios serán pacientes, médicos, familiares, etc.

En los últimos años, el número de redes sociales se ha incrementado cada vez más. Este fenómeno ha hecho que los estudios sobre redes sociales se incrementen y también sus diferentes implementaciones. En el caso de este proyecto, se busca una plataforma simple que ofrezca características básicas de comunicación y notificaciones. Además, esta debe ser de código libre y lo más adaptable posible, ya que la integración del sistema multi-agente puede requerir una gran modificación del código. Las opciones estudiadas que cumplen estos requisitos son:

- **BuddyPress** [52]: Sistema desarrollado por la comunidad del CMS¹ Wordpress. Está diseñado para ser altamente adaptable gracias a un sistema de componentes que permite activar o desactivar características, además de desarrollar nuevas

¹Sistema de gestión de contenidos

funcionalidades fácilmente. Además, dispone de una gran comunidad de la que se puede obtener soporte durante el desarrollo.

- **Elgg** [14]: Motor para desarrollar todo tipo de redes sociales bajo licencia GPL [25]. Las características más interesantes de esta plataforma son la API² para el desarrollo de *plugins* y la API de servicios Web, que podrían usarse para desarrollar la interfaz entre la red y los sistemas multi-agente de una forma más sencilla.

2.2 Sistemas multi-agente

Un sistema multi-agente [51] es un sistema basado en múltiples agentes inteligentes (con cierto nivel de autonomía). Los agentes se encuentran en un entorno concreto, del que captan y acumulan información para generar nuevo conocimiento y deliberar sobre sus acciones. La existencia de múltiples agentes hace que estos puedan comunicarse entre ellos, intercambiando conocimientos y servicios. El objetivo del sistema es solucionar problemas, que pueden ser tanto a nivel de agente como generales.

Al igual que en el caso de las redes sociales, ya se han estudiado en profundidad los sistemas multi-agente. En concreto, la fundación FIPA [21, 5] ha desarrollado un estándar que especifica cómo se deben comunicar los diferentes agentes en un sistema. Existen diferentes implementaciones que cumplen las especificaciones del estándar. De todas ellas, las opciones estudiadas han sido:

- **Jade** [31]: Framework implementado en Java. Incluye herramientas gráficas para mejorar la corrección de errores y el control en producción. Además, está diseñado para ser distribuido en múltiples máquinas de forma paralela.
- **SPADE** [36]: Plataforma implementada en Python. Los diferentes agentes se comunican entre ellos a través del protocolo XMPP³. Al igual que Jade, Spade es multi-plataforma y ofrece una interfaz para controlar el sistema.
- **Magentix 2** [32]: Plataforma implementada en Java. Está diseñada para funcionar en dominios reales tales como la industria, la logística o la medicina. Además, ofrece protocolos de interacción de gran flexibilidad y una gran seguridad.

En conclusión, ya hay mucho trabajo realizado en cuanto a sistemas multi-agente. Durante la realización del proyecto habrá que decidir qué plataforma se adapta mejor

²Especificación de la comunicación con un sistema informático

³Protocolo de comunicación basado en mensajes

a las características específicas del proyecto e implementar el sistema adaptándose a esta solución.

2.3 Integración de redes sociales y el sistema multi-agente

No se ha encontrado literatura sobre cómo integrar agentes en una red social moderna de cuidados médicos. Por esa razón, habrá que estudiar qué agentes integrar en el sistema, dónde integrarlos (cliente o servidor), qué interacciones habrá entre ellos, etc. No obstante, existen diferentes estudios que pueden ayudar en esta tarea. Por ejemplo, en [39] se estudian diferentes formas de aplicar tecnologías inteligentes para el cuidado de personas mayores. De forma más específica, en [28] se describe la arquitectura de un sistema multi-agente para respaldar comunidades, y en [46] se presentan diferentes características o funciones que podrían presentar los diferentes agentes.

En conclusión, existen diferentes estudios en los que basarse para desarrollar la solución, pero **hace falta un estudio profundo de determinados aspectos** del caso específico de este proyecto.

2.4 Servicios asistenciales

Durante el proyecto sólo se hará uso del caminador robótico i-Walker [1], por lo que la integración de las diferentes partes del sistema relacionadas con los servicios asistenciales se adaptará en menor o mayor medida a la implementación del caminador. Para los posibles futuros servicios asistenciales, podrían estar aquellos desarrollados en el seno de SHARE-it [13], proyecto en el cual está basado el i-Walker. No obstante, los sistemas estarán diseñados de la forma más dinámica posible, por lo que los desarrolladores podrán implementar servicios de todo tipo.

3

Red social

Se puede definir una red social [7] como una representación de una **estructura social en forma de grafo** $G = (V, E)$. Los vértices (V) de este grafo se conocen como actores sociales, y pueden estar conectados entre ellos (E) dependiendo de sus diferentes tipos de relación. Aunque los vértices más importantes en el grafo de la red son los individuos, también pueden representar otro tipo de concepto, tales como grupos o páginas.

Los actores de la red social pueden compartir información o crear nuevo contenido de forma pública o semi-pública, que puede ser de interés para los demás actores. Estos datos pueden ser accedidos por los demás dependiendo de su nivel de privacidad. La creación de contenido puede dar lugar a nuevos vértices (un informe de actividad, por ejemplo) y a nuevas aristas. Además, la estructura en forma de grafo permite a los actores obtener información que no esté directamente relacionada con ellos, ya que pueden navegar por el grafo siempre que la privacidad fijada lo permita.

Un ejemplo de vértices V en el grafo G pueden ser dos usuarios. Si los dos vértices no están conectados de ninguna manera, no pueden acceder a ningún contenido de la otra persona que no sea público. Sin embargo, podría ser que existiera una arista E que conectaría a las dos personas en una relación de amistad (podría ser otra). En este caso, podrían acceder al contenido semi-público ofrecido para este tipo de relación. Por último, también podría darse el caso de contenido accesible como máximo a dos niveles de profundidad. Es decir, contenido que puede ser accedido por usuarios directamente relacionados u otros usuarios que disponen de una conexión con alguna esas relaciones directas.

La imagen de la figura 3.1 muestra un ejemplo del grafo completo de una red social. En la imagen se pueden ver los vértices (círculos) conectados a otros vértices mediante aristas. Puede apreciarse también una característica común de gran importancia en la mayoría de redes sociales. Se trata de la aparición de grupos (o *clusters*) donde el número de relaciones entre actores dentro del grupo es mucho mayor que con actores de fuera de este. Este aspecto será muy importante a la hora de diseñar la red social de este proyecto y se verá en más detalle más adelante.

Este proyecto tiene el objetivo de desarrollar una red social médica. Esta red dispondrá de diferentes tipos de usuarios, que representarán a los tipos de personas

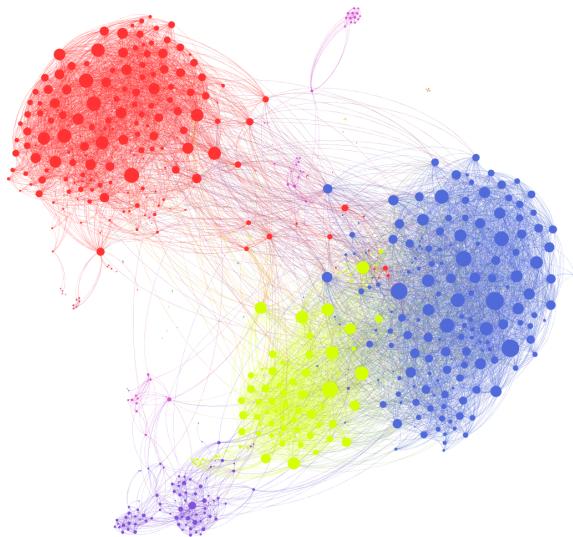


Figura 3.1: Ejemplo del grafo de una red social. Fuente: Griff's Graphs

presentes. Los usuarios que representen a los pacientes podrán disponer de dispositivos asistenciales asignados. Además, existirá una relación en la red social entre las personas que se encuentren en el mismo ámbito médico (paciente, médico, familiar, etc.).

En este capítulo se definen las características y el diseño del grafo de la red para el caso particular de este proyecto. Se empezarán definiendo los tipos de vértices y sus diferentes atributos. Después se explicitarán todas las relaciones que pueden existir entre ellos.

3.1 Vértices

Los actores sociales (vértices) dentro del grafo de la red estarán distribuidos en diferentes categorías. El primer grupo, y más importante, se compondrá de diferentes tipos de personas (roles). Además, también se fijarán vértices para los dispositivos asistenciales, para informes de actividad y mensajes entre usuarios. En las siguientes subsecciones se explica cada uno de estos tipos.

3.1.1 Personas

Las personas serán los vértices más importantes en la red. El grafo social simboliza una red social médica, por lo que habrá que representar a todas las personas interesadas en el proceso médico. Por tanto, los vértices de cada persona podrían cambiar de forma notable dependiendo de qué rol estén realizando. No obstante, las mayores diferencias serán visibles en las aristas de estos vértices.

En general, todos ellos dispondrán de la siguiente información:

- **Identificación en la vida real:** en el caso de España podría ser el DNI¹.
- **Identificador dentro de la red:** cada persona deberá disponer de un identificador en la red, que puede ser generado automáticamente a partir del identificador real.
- **Contraseña:** necesario para que el usuario pueda acceder a la red.
- **Nombre completo** (nombre y apellidos)
- **Correo electrónico** (si tuviera)

El diseño de estas características se ha fijado de una forma básica, para que un desarrollador que necesitara otros tipos de datos pudiera añadirlos más tarde. Por ejemplo, si se usa fuera de España, la identificación no sería el DNI, sino otro documento. Además de estas características, cada persona dispondrá de un rol. Este rol tendrá ciertas **características únicas** que lo diferencian del resto. Para este proyecto se han diseñado unos pocos roles, pero podrían ser extendidos en un futuro. Estos son:

1. **Paciente:** son los usuarios más importantes de la red social. Aunque no son los vértices con un mayor número de conexiones (son los del personal sanitario), sí son los vértices a través de los cuales se genera un mayor número de consultas. Esto hará que existan pequeños grupos muy centralizados en el paciente.
2. **Doctor:** son los encargados de dirigir el tratamiento de los pacientes. Por tanto, estarán conectados a los pacientes y podrán ver sus progresos o comunicarse con ellos o otros usuarios relacionados con él.
3. **Familiar:** personas con cierta relación personal con el paciente. Al igual que los doctores, podrán ver cierta información de sus pacientes relacionados y comunicarse con el personal sanitario que los atiende.
4. **Enfermero:** rol parecido al de los doctores, pero con menos privilegios.

¹Documento Nacional de Identidad

5. **Secretario:** se encarga de gestionar algunos aspectos de la red social. Su función es puramente administrativa, por lo que no forma parte directa de la red social. Esto implicará que su vértice estará aislado del de otras personas.

3.1.2 Dispositivos asistenciales

Los dispositivos tendrán dos estados excluyentes entre sí. Estos estados serán para indicar si el dispositivo está libre o asignado. Un dispositivo libre es aquel que no está siendo usado por ningún paciente en ese momento, mientras que uno asignado sí lo está. Cada paciente podrá tener un número indefinido de dispositivos asistenciales asignados. El personal sanitario será el encargado de asignar o desasignar un dispositivo a un paciente cuando sea necesario. Todos estos procesos se llevarán a cabo a través de la red a usando diferentes formularios. Algunos ejemplos pueden verse en el capítulo 10.

Un dispositivo podrá generar diferentes tipos de información, que podrá ser vista sólo por los usuarios relacionados por el paciente al que está asignado. Cada dispositivo dispondrá de tres campos de información básica y un campo de datos. Los campos son:

- **Identificador:** número de identificación único, que permitirá que el dispositivo se comunique con la red social, y esta relacione el contenido creado al paciente.
- **Tipo:** identificador del tipo de dispositivo. Cada tipo de dispositivo tendrá una cadena de identificación única para que el sistema pueda realizar acciones personalizadas dependiendo del dispositivo.
- **Clave:** al igual que las personas, un dispositivo deberá autenticarse en la red para poder enviar sus datos. La clave se comprobará cuando el dispositivo empiece a enviarlos.
- **Datos:** datos introducidos por el propio dispositivo para guardar información de su estado. Cada dispositivo podrá usar este campo como mejor le convenga.

3.1.3 Informes

El sistema multi-agente o el personal sanitario podrán generar informes relacionados a un paciente. Los informes podrán contener cualquier tipo de información relevante sobre el paciente. Los atributos de cada uno serán:

- **Resumen:** pequeño resumen del contenido del informe. Será el texto que se visualizará en el resumen de actividad del usuario.
- **Contenido:** informe completo.
- **Tipo:** existirán tres tipos diferentes de informes: información, aviso y alerta. Los informes catalogados como alerta generarán notificaciones y avisos para poder dar una solución rápida al problema.

3.1.4 Mensajes

Los usuarios de la red podrán enviar mensajes de texto a los demás usuarios relacionados de forma directa o indirecta. Por ejemplo, los familiares de los pacientes podrán enviar mensajes a estos, a otros familiares o al personal sanitario que los atiende. En este último caso, la relación no es directa, ya que el personal sanitario está relacionado sólo con el paciente, y a su vez el paciente con su familiar. El grafo de la red permite navegar por él para conseguir esta meta. Los mensajes dispondrán de un asunto y un contenido, e indicarán quién lo ha enviado y qué relación existe entre el remitente y el destinatario.

3.2 Relaciones entre vértices

Una vez definidos todos los vértices, el siguiente paso es relacionarlos entre ellos. De otra forma no se podrían explotar todas las características de una red social. Debido a que los vértices pueden ser o no personas, y además existen diferentes roles dentro de estas, también existirán multitud de relaciones. Por tanto, en las siguientes subsecciones se definirán las diferentes aristas V del grafo G de la red social. A partir de los vértices y las relaciones establecidas entre ellos se podrán implementar todas las características marcadas como un objetivo anteriormente. No obstante, ese tema forma parte de la implementación y no del diseño de la red, y se explica de forma detallada en el capítulo 6.

Un esquema de todas las relaciones posibles entre los vértices puede observarse en la figura 3.2. Este muestra una serie de vértices que podrían existir en una red ficticia. Los vértices están relacionados entre ellos, mostrando algunas de las características explicadas en las subsecciones siguientes.

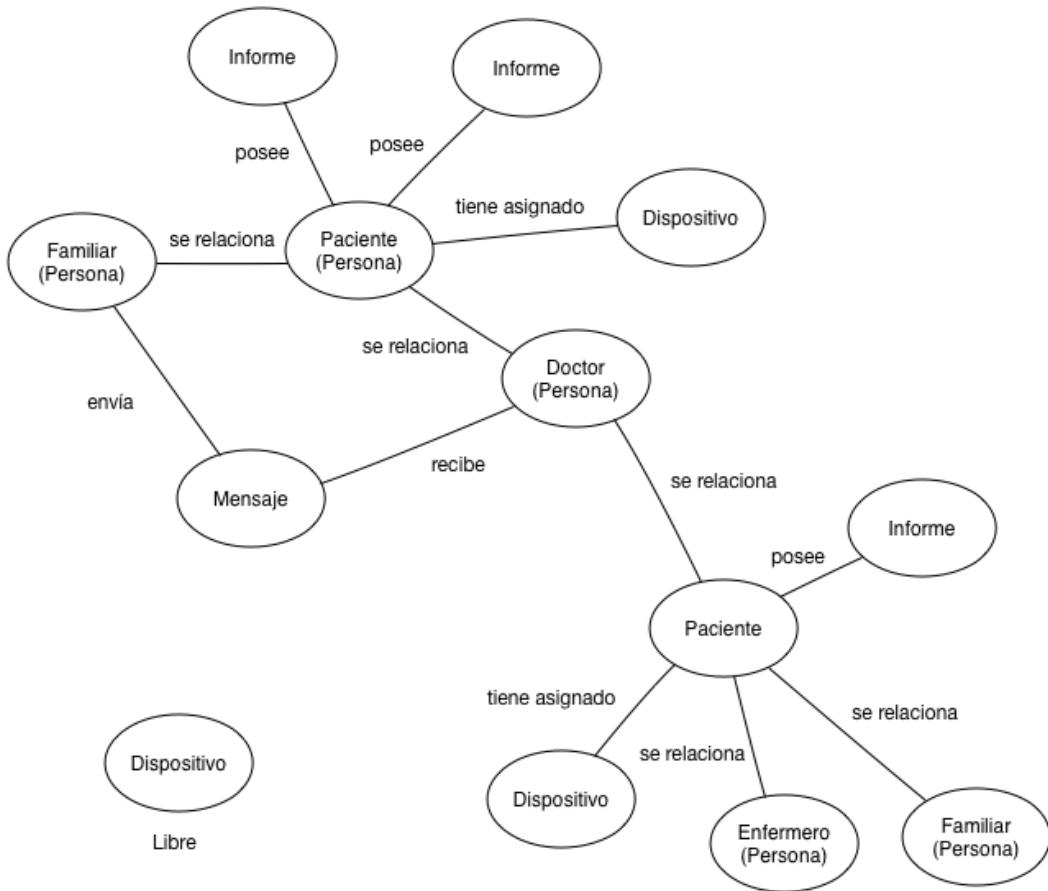


Figura 3.2: Ejemplo del diseño de los vértices de la red y sus relaciones

3.2.1 Relaciones entre usuarios

A diferencia de una red social clásica, donde sólo existe la relación de *amigos*, en la red social de este proyecto cualquier usuario no puede relacionarse con cualquier otro. Esto se debe a que cada usuario debe tener un rol fijado anteriormente. Los roles de los usuarios a relacionar, por tanto, deben ser compatibles. La compatibilidad entre ambos viene fijada por la consecución o no de alguno de los objetivos de la red. Es decir, ¿la relación refleja de forma coherente el ecosistema médico del paciente? Esto provoca que haya un número muy limitado de relaciones entre roles, incluso que algunos roles no puedan tener ningún tipo de relación.

Las relaciones, al igual que sucede en las redes clásicas, son bidireccionales. Por tanto, si un usuario está relacionado con otro, el otro lo estará de la misma forma pero en sentido contrario. Esto lleva a que, con los roles fijados anteriormente, sólo existan

tres tipos de relaciones entre vértices personales. Estas son:

1. Paciente y doctor
2. Paciente y familiar
3. Paciente y enfermero

Se puede apreciar como, de nuevo, los vértices en los que se centra el diseño son los pacientes. Podría ser posible relacionar usuarios de otras maneras diferentes. Por ejemplo, se podrían relacionar doctores y enfermeros (sin ningún paciente de por medio). No obstante, la red de este proyecto se centra en el desarrollo de dispositivos asistenciales, por lo que las relaciones deben incluir a un paciente (persona que puede tener alguno asignado). No obstante, sería posible que más adelante se incluyeran nuevos roles y funcionalidades que no tuvieran en cuenta a los pacientes.

Existen también otras relaciones no reflejadas que incluyen a los pacientes de forma indirecta, como por ejemplo un familiar y el doctor que atiende a su pariente. No obstante, el diseño de la red permite realizar estas relaciones navegando un poco por el grafo G (a través del vértice del paciente). De esta forma se ahorra en complejidad y no se pierde en velocidad, ya que la red es heterogénea y *clusterizada* [24]. Esto se debe a que, como se ha visto ya, está formada por grupos muy excluyentes entre sí. De esta manera, es fácil navegar al resto de vértices del grupo una vez se sabe cuál es el vértice del paciente. Por ejemplo, si un familiar quiere mandar un mensaje a un enfermero de su pariente, el sistema sólo tendrá que navegar primero al vértice del paciente y entonces acceder a los vértices de los enfermeros relacionados con este. Otro ejemplo de esto puede ser relacionar a un doctor y un enfermero (ejemplo de relación no presente visto en el párrafo anterior). La diferencia en que en este caso ambos deben estar atendiendo al mismo paciente. Es decir, no es una relación directa. De hecho, un mismo doctor y enfermero podrían estar relacionados indirectamente dos veces con pacientes diferentes.

Estos pequeños grupos de pacientes, como se ha comentado, son altamente excluyentes entre ellos. Los pacientes, de hecho, sólo podrán tener relaciones dentro de su propio grupo. Los familiares, y en mayor medida los médicos y enfermeros, podrían tendrán conexión con más de un grupo a la vez. Esto se debe, por ejemplo, a que un médico puede estar atendiendo a varios pacientes a la vez. No obstante, las acciones dentro de la red deben estar siempre limitadas a un único grupo. De otra manera, un usuario podría acceder a datos de otros usuarios con los que no tiene relación directa o lógica. La existencia de estos grupos es una decisión de diseño en este proyecto. Sin embargo, podría ser que en un futuro se quisiera relacionar pacientes entre ellos o nuevos tipos de relaciones entre otros roles.

Como es apreciable, el rol de secretario no puede relacionarse con más personas. El

vértice de este tipo de usuarios, por tanto, no estará conectado al de ninguna otra persona. No obstante, **tendrá poder para modificar los vértices de los demás usuarios** y añadirles ciertas aristas. Por ejemplo, un secretario puede relacionar a dos personas, o asignar dispositivos asistenciales a un paciente.

3.2.2 Relación entre pacientes y dispositivos

Un dispositivo asistencial puede estar asignado a más de una persona durante su vida útil. Por tanto, es posible que la relación que los une cambie en un momento dado. En cualquier momento, no obstante, **un dispositivo asistencial tendrá una única relación** con otro vértice. Esta relación marcará el usuario al que está asignado en ese momento. En caso de que se asigne a otra persona, primero se deberá eliminar la relación anterior y, después, añadir el mismo tipo de relación en el otro paciente. Esta es una decisión de diseño y podría cambiarse para ser más complejo. Por ejemplo, podría fijarse que un dispositivo está asignado a diferentes personas dependiendo del día de la semana, que podrían estar haciendo rehabilitación sólo esos días. No obstante, este aspecto queda fuera del alcance del proyecto. Además, cabe decir que un dispositivo sólo puede estar relacionado con aquellos usuarios que dispongan del rol de paciente, y no con otro tipo de usuarios.

Un dispositivo, por tanto, puede tener dos tipos de estado. El primero de ellos es el estado de *asignado*, que obtiene cuando dispone de una relación con un paciente. El segundo estado es el de *libre*. En este último estado, el dispositivo se muestra activo para ser asignado a otro paciente.

3.2.3 Otras relaciones

Algunas relaciones entre vértices pueden tratarse como una posesión. Por ejemplo, si un usuario recibe una notificación, la relación que los une es de posesión. Por tanto, tiene el poder de eliminarla. Esto también puede darse cuando un usuario publica un comentario o, por ejemplo, cuando el sistema guarda un registro de las acciones de un usuario.

Otro tipo de relación puede incluir una creación. Por ejemplo, un dispositivo o una persona relacionada pueden componer un informe de un paciente. El paciente puede obtener todos los informes que se han creado en referencia a su persona, pero no puede eliminarlo ya que no es su creador. Por tanto, un informe tendrá dos relaciones principales. La primera de ellas lo relacionará con el paciente al que va dirigido. La segunda lo relacionará con el creador de dicho informe, que puede ser tanto otra

persona como un dispositivo asistencial que ha generado el informe a través del sistema multi-agente.

Existen otros vértices que tienen una sola relación con dos o más personas. Un ejemplo de este tipo de relación se encuentra en los mensajes. Los mensajes están relacionados, primero, con la persona que lo envió, y también con todas las personas que recibieron el mensaje. Este tipo de relaciones, sin embargo, se suelen implementar de diferentes formas para ahorrar complejidad. Aún así, esa relación se ha diseñado de esa manera.

4

Sistema multi-agente

Un agente [51] es un sistema informático capaz de un cierto nivel de autonomía, el cual usa para descubrir qué acciones debe realizar para satisfacer unos objetivos sin necesidad de órdenes precisas. La complejidad de un agente puede variar desde agentes totalmente autónomos hasta entidades reactivas simples (listado de reglas a realizar ante ciertos eventos).

Un sistema multi-agente [50] (también conocido como MAS) es cualquier sistema compuesto por múltiples de estos agentes que interactúan entre ellos. Para que la interacción sea satisfactoria, los agentes deben tener la capacidad de cooperar, coordinarse y negociar entre ellos.

Los agentes están presentes en un entorno, en el cual pueden captar información y realizar acciones (figura 4.1). La información recibida del entorno (conocimiento) puede ser acumulada por los agentes como se crea conveniente. A partir de este conocimiento, el agente podrá decidir las acciones o tareas a realizar a continuación dependiendo del estado del entorno y de sus propios objetivos. Las acciones pueden ser muy variadas, desde generar un plan para lograr un objetivo concreto hasta eliminar algunos de los objetivos que disponga en ese momento.

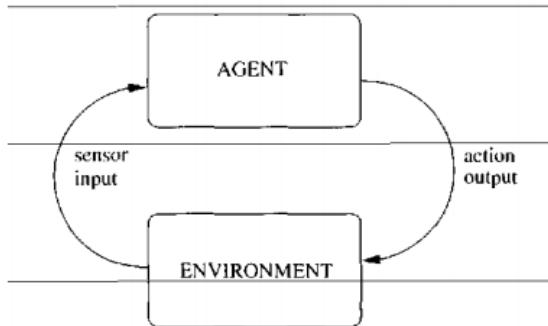


Figura 4.1: Esquema de un agente en su entorno. El agente capta información del entorno y produce acciones que afectan a este. Fuente: [50]

Los entornos donde se encuentran los agentes pueden tener características diferentes, haciendo variar su complejidad. Los agentes deben actuar en el entorno en tiempo real, por lo que estas características deben ser estudiadas en profundidad. A continua-

ción se listan diferentes tipos de entorno separados por características que podrían considerarse deseables:

- **Accesible:** entorno en el cual es posible obtener información de forma completa, precisa y actualizada.
- **Determinista:** una acción concreta tiene siempre el mismo efecto sobre el estado del entorno. No hay dudas sobre cómo se encontrará después de realizar una acción.
- **Estático:** el estado del entorno no cambia excepto por la acción del agente.
- **Discreto:** sólo se pueden realizar un número fijado y finito de acciones sobre el entorno.

La mayoría de los entornos en la vida real no cumplen completamente las características descritas anteriormente. Por esta razón, el diseño de los agentes debe tener en cuenta estos factores a la hora de captar datos (pueden no ser precisos en un entorno inaccesible), realizar sus acciones (pueden dar lugar a un estado inesperado en un entorno no determinista), etc.

Para alcanzar sus objetivos, los agentes podrán (además de realizar sus propias tareas) comunicarse y cooperar con otros agentes en su entorno. Esta cooperación puede llevarse a cabo compartiendo conocimiento o intercambiando otro tipo de acciones. Para facilitar la cooperación, normalmente los demás agentes se consideran veraces y benevolentes, aunque esto puede no ser así. Por otro lado, para poder llevar a cabo esta cooperación, los agentes deben entenderse usando algún lenguaje de comunicación común.

En referente a este proyecto, el MAS tiene el objetivo de gestionar la información generada por los dispositivos asistenciales (ambiente), procesarla y compartirla con los demás agentes (red social de agentes) cuando se crea necesario. Además, podrá enviar datos de forma proactiva a la red social para que los usuarios reales puedan informarse.

Por otro lado, su diseño debe permitir dos aspectos fundamentales: funcionalidad y extensibilidad. Esto es, debe ser capaz de extenderse mediante nuevos módulos sin perder funcionalidades. En cada una de las secciones de este capítulo se detalla como conseguir estos aspectos mediante diferentes técnicas, esquemas y modelos.

4.1 Modelo de software

El primer paso que hay que llevar a cabo es decidir qué modelo de software se aplica a los agentes. Existen multitud de modelos que pueden usarse. No obstante, el sistema que se está diseñando tiene un aspecto muy esclarecedor, sus agentes más importantes representan personas. Por tanto, lo mejor es usar un sistema de razonamiento basado en el comportamiento humano. **Michael Bratman** [8] definió el razonamiento práctico de los humanos con tres aspectos fundamentales: creencias (*beliefs*), deseos (*desires*) e intenciones (*intentions*). Este modelo, a partir de ahora BDI, explica los procesos que cualquier ser humano realiza al querer conseguir algo (razonamiento práctico). Por tanto, es un buen modelo por el que empezar a diseñar el sistema.

Además de su estudio en psicología, el modelo BDI se ha estudiado en profundidad aplicado a las ciencias de la computación. En [41], Anald S. Rao y Michael P. Georgeff explican cómo aplicarlo a un sistema de agentes inteligentes para que, mediante la consecución de sus objetivos particulares, consigan satisfacer un objetivo común y general.

4.1.1 BDI

El objetivo principal de la arquitectura BDI es simple, conseguir ciertos objetivos. Dentro de este modelo, estos objetivos se conocen como deseos, es decir, algo al que el agente querría conseguir en algún momento futuro. Para llevar a cabo esos deseos, el agente debe tener un **conocimiento** más o menos preciso del entorno en el que se encuentra. Debido a que un entorno real no es determinista, el modelo BDI habla de creencias. Esto significa que un agente puede creer que algo es falso cuando en realidad es verdadero, y viceversa. El último factor que afecta al modelo son las intenciones. Las intenciones mezclan los dos conceptos introducidos anteriormente. Esto es, ¿qué va a hacer el agente dependiendo del estado de su entorno y sus deseos? Esto es importante, ya que un mismo agente puede tener diferentes deseos, cuyas consecuencias pueden ser excluyentes en un momento dado del tiempo (perseguir un objetivo implica no conseguir el otro de momento). El agente, por tanto, debe ser capaz de deliberar sobre qué deseo decantarse (si son excluyentes) y qué plan escoger.

4.2 Distribución de los agentes

Fijado el modelo para los agentes, hay que especificar qué tipos de agentes existirán y en qué número. De nuevo, el modelo BDI ofrece ciertas facilidades por ser un modelo basado en el razonamiento práctico humano. En este caso, existe una separación clara entre los grupos. Las subsecciones que siguen a continuación detallan cada grupo presente en el diseño.

4.2.1 Dispositivos asistenciales

Los dispositivos no representan personas, sino aparatos con sensores asociados a un paciente específico. Otra característica de estos agentes será el rango de complejidad, que se moverá desde agentes muy sencillos hasta muy complejos. Además, en este proyecto se busca dejar este aspecto abierto a nuevos desarrollos. Por estas razones, el diseño del MAS sólo ofrecerá un modelo de agente que representará a un dispositivo.

El agente del dispositivo ofrecerá las **características básicas para comunicarse con el sistema del paciente**. Es decir, toda la comunicación entre paciente y dispositivo deberá realizarse mediante este agente. No obstante, el resto queda abierto a los desarrolladores, por lo que será posible que este agente de comunicación reciba los datos de un subsistema de agentes que hayan procesado ya cierta información, o que este obtenga información directamente de los sensores.

4.2.2 Personas

Los agentes personales obedecen de una forma más precisa al modelo BDI. Por tanto, en este diseño se ha fijado un único agente por persona. Dentro de estos agentes, existirán diferentes tipos dependiendo de su rol, que tendrán comportamientos distintos.

Estos agentes deben estar abiertos a modificaciones al añadir nuevos dispositivos. Para ello, el paradigma de agentes ofrece facilidades. En lugar de añadir nuevos agentes para procesar esa información, los desarrolladores podrán introducir nuevos comportamientos que correspondan a un dispositivo. Por tanto, al crear un nuevo dispositivo se tendrá que fijar, además del agente del dispositivo, una **lista de comportamientos** que se añadirán a los diferentes agentes relacionados. Estos comportamientos podrán integrarse de forma independiente dependiendo de los agentes. De esta forma, un dispositivo podría incluir comportamientos sólo al paciente

que lo está utilizando o también a todos o algunos de los usuarios relacionados con el paciente. Usando estos nuevos comportamientos, los agentes podrán variar sus creencias, deseos e intenciones. Con este sistema se consigue una arquitectura más robusta y dinámica, que permite realizar grandes cambios sin necesidad de modificar el núcleo de los sistemas.

4.2.3 Gestor de agentes

Todos los agentes de objetos físicos se han fijado en este punto. El principal problema ahora es que dichos agentes no tienen conocimiento del resto de agentes que existen y cuál es su relación con ellos. En el estándar fijado por FIPA se especifican dos agentes obligatorios en cualquier plataforma de agentes [22], que se encargan de gestionar algunos aspectos de la comunicación entre agentes. En primer lugar se encuentra el DF (*Directory Facilitator*). Este agente se encarga de **gestionar los servicios** que ofrecen los diferentes agentes, para que así otros puedan encontrarlos y comunicarse con estos agentes. En segundo lugar se encuentra el AMS (*Agent Management System*), que se encarga de **gestionar el acceso y la presencia** de agentes en la plataforma. Cualquier agente debe registrarse en el AMS de la plataforma, momento en el cual recibirá un identificador AID¹, que servirá para poder comunicarse. El AMS permite, además, buscar a otros agentes sin tener en cuenta sus servicios.

En el caso de este proyecto los agentes están dispuestos de una forma muy concreta y delimitada. Esta limitación viene dada por la estructura de la red social. En concreto, cada usuario en la red social se ve representado por un agente que, en principio, sólo debería comunicarse con aquellos otros con los que disponga alguna relación (directa o indirecta). Por tanto, el sistema estará formado por grupos de agentes con un paciente, sus relaciones en la red social y los dispositivos que tenga asignados. Hay que tener en cuenta que un agente podría estar en más de un grupo a la vez, por lo que un grupo sólo indica los agentes con los que deberían comunicarse los demás agentes. Un ejemplo de esto puede ser un paciente, que sabe que puede comunicarse con sus familiares, con el personal sanitario y con los agentes de sus dispositivos. No obstante, nada limita al agente comunicarse con otros.

Como ya se ha comentado, algunos de los agentes pueden pertenecer a multitud de grupos a la vez. Por ejemplo, un médico puede estar atendiendo a varios pacientes a la vez. Sólo los pacientes y dispositivos tienen restringido la pertenencia a un solo grupo. Otro ejemplo de este fenómeno puede darse con los agentes de un familiar, que puede tener a diferentes parientes usando dispositivos.

¹Identificador del Agente (*Agent Identifier*)

El problema, pues, consiste en que los agentes no conocen qué otros agentes están presentes en sus grupos. Además, podría suceder que un agente entre o abandone un grupo. Por tanto, se hace necesario un agente que se encargue de **gestionar los diferentes grupos e informar a los demás sobre la presencia** de otros agentes. El agente gestor tendrá ese objetivo. Dispondrá un listado de agentes en el sistema (igual que el AMS de FIPA) y podrá ser preguntado por los demás agentes. Por ejemplo, un agente podrá preguntar sobre qué médicos hay presentes en sus grupos. Este aspecto guarda cierto parecido al ofrecido por el *directory facilitator* de FIPA, ya que los diferentes tipos de agentes ofrecerán diferentes tipos de funciones, aunque de una forma más limitada del fijado en el estándar.

La comunicación entre agentes, no obstante, no está limitada a los diferentes grupos. Limitar este aspecto dependerá de cada agente, que podrá preguntar si el agente que ha enviado el mensaje que acaba de recibir pertenece a alguno de sus grupo o no. Esto se fijado de esta manera ya que no queda cerrada la introducción de más tipos de agente, cosa que limitaría la comunicación. Además, el agente gestor no sustituye al AMS o DF. En primer lugar porque, para ser registrado en el gestor, un agente debe tener ya un AID correcto, es decir, debe estar ya registrado en el AMS. En segundo lugar porque no gestiona a todos los agentes de la plataforma, solo aquellos que tengan alguna relación con un paciente. Por tanto, si se añaden nuevos tipos agentes en un futuro proyecto, estos no serán gestionados por él. Por último, como la comunicación entre agentes se ha delimitado casi de forma estricta a los grupos cerrados, no existen servicios como tal. Por contra, el rol del agente delimita en gran medida la comunicación con los demás agentes. No obstante, el DF podría ser usado normalmente por cualquiera de los agentes.

Otro aspecto importante en el sistema es que los agentes de los dispositivos no se ejecutan en la misma máquina que el resto del sistema. Esto significa que el gestor no podrá controlarlos directamente, ni ofrecerles los datos necesarios a la hora de iniciar el agente. Por tanto, este agente también servirá como iniciador de los dispositivos. Para ello, el dispositivo enviará una consulta al gestor, indicándole que va a iniciarse. Junto a esta consulta, enviará su identificador y su clave. Primero, el gestor comprobará que los datos identificativos son correctos y que el dispositivo se encuentra en la lista de dispositivos asignados a algún paciente. El gestor dispondrá de todos los datos necesarios, exceptuando la dirección que identifica al agente del dispositivo. En ese momento, pues, guardará la información que le faltaba y enviará los datos necesarios (que el agente del dispositivo no conoce) para que este pueda empezar su actividad. El más importante de estos datos es la dirección del agente del paciente, ya que el dispositivo no conoce a quién está asignado.

4.3 Arquitectura

Definido todo lo anterior, solo resta saber cómo organizar el sistema para que sea escalable. Al igual que en la sección anterior, el esquema más natural suele ser también la mejor opción. Para entender mejor el diseño, se ha separado la explicación en diferentes subsecciones, una para cada nivel natural del sistema. Estos niveles son: arquitectura base, agentes personales, dispositivos asistenciales y gestor de agentes. Un esquema de toda la arquitectura se puede ver en la figura 4.2.

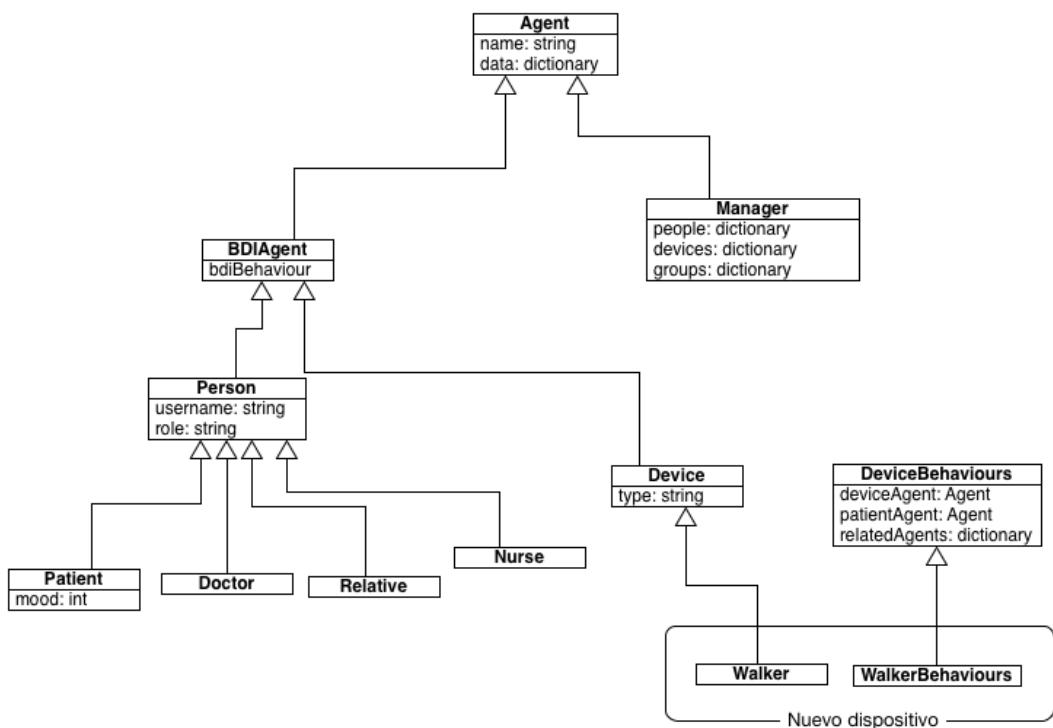


Figura 4.2: Arquitectura del sistema multi-agente

4.3.1 Base

El primer aspecto de la arquitectura es una clase de agente del que heredarán el resto. Este agente toma el nombre de **Agent**. El agente base sólo dispondrá de los datos básicos, es decir, un nombre y una variable para almacenar datos. Además, dispondrá de las acciones básicas para comunicarse con otros agentes. Estas funciones incluyen responder a un mensaje dado o esperar una respuesta a un mensaje enviado.

Del agente **Agent** heredará otro agente genérico, que será el encargado de gestionar las creencias, objetivos e intenciones de todos aquellos agentes que hereden de él. Este agente, que tomará el nombre de **BDIAgent** tendrá las mismas variables que el agente base, pero dispondrá de un comportamiento periódico para gestionar el modelo BDI. Este comportamiento se ejecutará cada cierto tiempo, y durante su ejecución comprobará si los objetivos activos se han cumplido, activará los nuevos, ejecutara normas (tareas a realizar dependiendo de unas condiciones) y deliberará sobre las acciones a realizar en un futuro.

4.3.2 Agentes personales

Cada rol en la red social dispondrá de una clase de agentes, que heredarán de **BDIAgent**. En este diseño se tienen en cuenta cuatro roles diferentes, aunque se puede extender si fuera necesario. Estos roles son: paciente, doctor, familiar y enfermero. De ellos, el rol de paciente es el más complejo y el que requiere un mayor desarrollo. En las siguientes subsecciones se detalla el diseño de cada uno de estos agentes.

Paciente

El paciente es el agente más importante de entre los agentes personales. Su objetivo consistirá en gestionar el estado del paciente, recibir datos de los dispositivos asistenciales y comunicar aquellos importantes tanto a la red social como a los demás agentes relacionados. El principal problema que surge a la hora de diseñar este agente es saber qué conocimientos y funcionalidades puede albergar. Este problema surge porque los datos que recibirá de los dispositivos que tenga asignados puede ser diversos y de una gran diferencia entre ellos. Por tanto, deberá procesarse de diferentes formas, algo que tendrá que definir cada uno de ellos. Este aspecto se ve en más detalle en la sección 4.3.3, que explica el diseño de los dispositivos asistenciales.

Por tanto, el principal problema del diseño es **fijar qué conocimientos y funcionalidades** pueden querer compartir todos los dispositivos de un paciente. Esto hace volver a observar los objetivos principales del proyecto, que se basan en mantener informados a todos los actores relacionados con el paciente para mejorar su estado de ánimo y tratamiento. Por tanto, teniendo en cuenta estos datos, se puede fijar el conocimiento y funcionalidades del agente en los siguientes puntos:

1. El agente dispondrá de una estimación del estado de ánimo del paciente (en un rango acotado). Los diferentes dispositivos podrán acceder a este valor

para consultar esta estimación, y modificarla como crean dependiendo de las acciones que realice el paciente en el dispositivo que controlan.

2. El agente guardará conocimiento de posibles peligros sucedidos en alguno de los dispositivos y ofrecerá métodos para alertar a los demás agentes relacionados. Debido a que la definición de peligro puede ser diferente para cada tipo de dispositivo, la responsabilidad de definirlos recae sobre cada uno de ellos. Es decir, el agente del paciente sólo ofrecerá herramientas para informar a los demás agentes sobre los peligros, pero no para explicitar cómo suceden.

Los peligros deberán ser definidos en los comportamientos que los dispositivos añadan al paciente (sección 4.3.3). Un peligro se representará usando el conocimiento del agente del paciente, indicando de qué tipo es (tipo de dispositivo y nombre del peligro) y el tiempo en el que se ha captado. Un peligro podría ser definido de dos formas. La primera forma es a través de la lectura directa de los sensores, fijando límites para algunos de los datos. La segunda se basa en definirlos a través implicaciones lógicas del conocimiento anterior. Por ejemplo, un agente podría guardar constancia de ciertos eventos y decidir que existe un peligro si un subgrupo de ellos ha sucedido en un periodo de tiempo determinado. De esta forma, podría generarse un peligro dados dos eventos en tiempos diferentes que por sí solos no representan ningún peligro.

3. El agente ofrecerá funcionalidades para informar sobre algún aspecto de un dispositivo a un grupo determinado de agentes. Esta información puede ser de cualquier tipo, por lo que los demás agentes deberán saber descifrar el mensaje.

En resumen, el **diseño del agente del paciente es muy simple**. Sólo posee conocimientos y funciones que puedan usar todos los nuevos dispositivos para realizar sus objetivos propios. Por tanto, la mayoría de la responsabilidad no recae en el propio agente, sino en los comportamientos que añadirán los dispositivos, que deberán definir comportamientos para cada tipo de agente. Estos nuevos comportamientos se añadirán cuando el dispositivo se inicie, y harán posible que los agentes puedan procesar sus datos. Por ejemplo, el agente de un dispositivo puede estar obteniendo datos de sus sensores y enviarlos al paciente. El problema es que el paciente no sabe cómo procesarlos. Este dispositivo podría definir nuevos comportamientos para su agente, para que pudiera ser capaz de procesar los datos que le envía. Podría ser necesario también comunicar estos datos procesados con los demás agentes, por lo que el dispositivo también tendría que añadirles nuevos comportamientos.

Con este diseño se permite un comportamiento mucho más dinámico pero a la vez menos controlado. Este aspecto tiene ventajas e inconvenientes, pero el dinamismo es uno de los principales objetivos de este proyecto en cuanto a dispositivos asistenciales. Por esa razón se cree que responsabilizar a los dispositivos de los comportamientos

de los demás agentes a la hora de procesar sus datos es la mejor opción. No obstante, estudiar mejoras para el agente del paciente es posiblemente una de las mejores ideas para futuros proyectos.

Doctor, familiar y enfermero

El mayor peso del trabajo recae en el agente del paciente. No obstante, deben existir también agentes para las personas relacionadas con él. Todos estos agentes recibirán datos del agente del paciente, y deberán decidir qué hacer con ellos. A diferencia del paciente, que tiene diversas necesidades, la única necesidad básica de estos agentes es tener informados a las personas que representan sobre el estado de su paciente relacionado.

La necesidad de estar informados, pues, no cambia dependiendo del rol. Por tanto, todos estos agentes dispondrán de un comportamiento en el cual esperarán mensajes de alguno de sus pacientes relacionados. Cuando los reciban, procesarán la información dependiendo de tres factores:

1. **Peligro:** el agente de paciente enviará un mensaje a todos sus agentes relacionados si estima que este está sufriendo algún tipo de peligro. Por ejemplo, podría informar si cree que el estado de ánimo de este está demasiado bajo, o si se han detectado demasiados avisos en alguno de sus dispositivos.
2. **Alerta:** se avisará a los agentes relacionados si un dispositivo produce alguna alerta. En el caso del caminador i-Walker, por ejemplo, se produciría una alerta si se detecta una posible caída.
3. **Nuevo ejercicio o actividad:** el paciente informará cuando un dispositivo envíe un análisis de sus datos. Esto sucederá sólo si el agente del dispositivo cree necesario informar a los demás. Dependiendo de estos datos, el agente podría proceder a informar a la red social, o podría ignorarlo. Hay que tener en cuenta que las diferentes actividades o ejercicios aparecerán en la lista de actividad de la red, aunque el agente no decida informar sobre ello. Esto se debe a que será el agente del paciente el que genere el informe, y la red mostrará todos los informes de paciente relacionadas.

Por tanto, estos agentes ofrecerán herramientas básicas para realizar estas funciones, pero no serán complejos. La razón es que algunos de estos aspectos dependen en gran medida del tipo de dispositivo que esté generando los datos. Funcionalidades más complejas, sin embargo, pueden ser añadidas a través de los comportamientos insertados al agente por un dispositivo concreto, que podrá procesar sus datos de diferentes formas (este aspecto se detalla en la siguiente sección). Por otro lado, estos

tres tipos de agente podrían ver aumentadas sus capacidades en futuras revisiones donde se añadan otros tipos de servicios no relacionados con los vistos en este proyecto.

4.3.3 Dispositivos asistenciales

Los dispositivos asistenciales dispondrán de dos partes diferentes. Por un lado se encuentra el agente del dispositivo y por otro lado los comportamientos a añadir a sus agentes relacionados. En el primer caso, existirá una clase de nombre `Device` del que deberán heredar todos los agentes de dispositivos nuevos. Esta clase, además de los datos de su superclase, incluirá una variable para indicar el tipo de dispositivo. También ofrecerá herramientas para facilitar el trabajo a los desarrolladores para comunicarse directamente con el paciente al que el dispositivo está asignado, y también para comunicarse con el gestor de agentes. Hay que tener en cuenta que el agente del dispositivo estará ejecutándose en el propio dispositivo y no en el servidor (donde se encuentran los agentes personales). Por tanto, un dispositivo tendrá que comunicarse con el gestor antes de poder iniciar su actividad completa. Esta comunicación le proporcionará los datos del agente al que el dispositivo está asignado. De otra forma, el dispositivo no sabría a quién enviar los datos procesados.

Por otro lado, los nuevos dispositivos tendrán que especificar unos comportamientos que serán añadidos a algunos (o todos) de los agentes relacionados con el dispositivo. Estos añadirán las funcionalidades necesarias para tratar con los datos generados por el dispositivo, las cuales no disponían anteriormente.

A diferencia del agente del dispositivo, que se ejecuta en su propia máquina, los comportamientos se añadirán a los agentes que ya están presentes en el servidor. Una plantilla para añadirlos más fácilmente estará incluida en la clase `DeviceBehaviours`, que cada nuevo dispositivo tendrá que heredar y personalizar. Esta plantilla recibirá como parámetros el agente del paciente y los agentes de las personas relacionadas con él. De nuevo, hay que tener en cuenta que este proceso se lleva a cabo del lado del servidor, máquina donde no se encuentra el agente del dispositivo. Será el gestor de agentes (sección 4.3.4) el que se encargue de añadir los nuevos comportamientos usando la plantilla del dispositivo. Esta tarea se llevará a cabo una vez que el agente del dispositivo haya finalizado su configuración con el gestor (explicada en detalle en la sección 7.3.4). El desarrollador deberá añadir aquellos comportamientos necesarios a cada agente (dependiendo de su rol) que permitan procesar los datos del dispositivo. Debido a que estos comportamientos puedes tener una complejidad elevada o variar con el tiempo, es el desarrollador el que debe encargarse de llevar un control de estos (controlar si han finalizado, etc.).

Hay que tener en cuenta que el agente del dispositivo y los comportamientos añadidos a los agentes se estarán ejecutando en sitios diferentes. Por un lado, el agente se ejecutará en el dispositivo. Por otro lado, sus comportamientos en el servidor. Por eso, se puede dividir el procesado de datos del dispositivo en las dos partes. La primera de ellas puede realizar tareas sin un gran impacto procedural, de las cuales enviará el resultado al servidor, que realizará un procesado mayor. De esta manera se aumentará la autonomía de los dispositivos. Además, en el servidor se posee de más información sobre el paciente, que puede procesar datos de diferentes dispositivos a la vez.

Algunos de los dispositivos, no obstante, pueden generar una gran cantidad de datos. Todos estos datos deberían enviarse a través de la red hacia el servidor para su posterior procesado. En este caso, sería probable realizar un mayor procesado en el propio dispositivo para que no se forme un cuello de botella en el envío. Por tanto, el trabajo a realizar en cada una de las partes no se especifica de forma estricta, sino que dependerá de las características o situación particular de cada dispositivo.

4.3.4 Gestor de agentes

El gestor de agentes es una de las partes más importantes de la arquitectura del sistema. Su función consiste en **harmonizar el resto del sistema**, añadiendo la posibilidad de añadir nuevos agentes, eliminar otros, gestionar los grupos de pacientes o informar a los que ya existen sobre los demás. Es decir, el gestor es el agente que hace de interlocutor entre las acciones producidas en la red social y el resto de agentes. El gestor guarda toda la información de los demás agentes y sus relaciones en la red. Por tanto, los agentes deben estar administrados por este gestor. En caso de que no lo estuvieran, podrían funcionar de la misma forma, pero se perdería la sincronización con la red social.

El gestor tiene dos partes fundamentales bien diferenciadas: la gestión de los agentes y las consultas que puede gestionar. En las siguientes subsecciones se detalla el diseño de cada una de ellas y las posibilidades que ofrecerán.

Gestión de agentes

Los diferentes agentes de la red social deben estar organizados en pequeños grupos, tal como se explicó en la sección 4.2.3. Estos grupos se componen de las relaciones que existen en la red social, y están centradas en los pacientes con dispositivos. Más en detalle, un grupo puede identificarse por el mismo identificador de un paciente.

Un grupo, por tanto, se compone de al menos un paciente, por lo que no puede existir un grupo que no disponga de paciente. Este grupo, además, puede contar con diferentes personas de roles diferentes, que disponen de algún tipo de relación con el paciente que lidera el grupo. Además de estas personas, el grupo contiene información sobre los dispositivos asignados a cada paciente. Una esquema de estos grupos se puede encontrar en la parte izquierda de la figura 4.3.

El gestor debe ser capaz de organizar todos los grupos existentes en el sistema. Cuando se inician los sistemas o hay cambios en la red social en tiempo de ejecución, el agente gestor debe ser informado para que haga los cambios necesarios en los diferentes grupos. De otra manera, podrían producirse inconsistencias o errores.

Un punto a tener en cuenta en estos grupos es que algunos de los agentes pueden estar dispuestos en diferentes grupos a la vez. Por ejemplo, un médico puede estar tratando a diferentes pacientes a la vez. Por esa razón, estará en diferentes grupos. Para evitar problemas y mejorar la facilidad de gestión, el gestor dispondrá de dos tipos de listas. El primer tipo dispondrá de los agentes en sí. De nuevo, para ganar expresividad, el gestor dispondrá de dos listas de este tipo. Por un lado, un listado de todos los agentes personales y, por otro, un listado con los dispositivos. El otro tipo de lista contendrá sólo referencias (mediante un identificador) a los agentes del primer tipo de listas. De este tipo sólo existirá un listado, que contendrá los diferentes grupos. Este esquema puede apreciarse más fácilmente en la figura 4.3.

El gestor es el interlocutor entre la red social y el resto de agentes. Esto significa que pueden darse cambios en la red que deben verse reflejados en los agentes. Por tanto, el gestor deberá gestionar los cambios obedeciendo las órdenes que lleguen desde la red social a través de la interfaz entre sistemas. Esta gestión de los agentes se basará en seis acciones diferentes:

1. **Crear un grupo** (añadir un nuevo paciente): es una acción fácil de realizar. En primer lugar, el gestor tendrá que comprobar si el paciente ya existe en el sistema. En caso de existir producirá un código de error. Si no existía antes, añadirá la persona al listado de agentes personales y creará un grupo vacío, al que añadirá el paciente.
2. **Agregar una persona a un grupo**: es posible que se cree una nueva relación entre un paciente y otra persona. En este caso, el gestor deberá también comprobar primero si el agente ya existe. En caso contrario, añadiría el agente al listado correspondiente. Después, en ambos casos, añadiría el identificador de ese agente al grupo especificado, en la sección de agentes relacionados.
3. **Eliminar una persona de un grupo**: por ejemplo, si se ha eliminado la relación entre el paciente y la persona. En este caso, el gestor primero com-

probará que el agente exista y se encuentre dentro del grupo especificado. Si todo fuera correcto, se eliminará la referencia en ese grupo en concreto. Hay que tener en cuenta que esta acción nunca elimina el agente de la lista, sólo referencias. Esto sucede aunque dicho agente ya no esté presente en más grupos.

4. **Eliminar una persona completamente:** puede suceder si se elimina una persona de la red social. Este caso dispone de dos fases. La primera de ellas consiste en eliminar las referencias del agente de todos los grupos en los que se encuentra. La segunda fase es eliminarlo del listado de agentes.
5. **Añadir/eliminar un dispositivo:** los dispositivos pueden ser asignados o desasignados a los pacientes de forma dinámica. A diferencia de las personas, si un dispositivo existe en la lista, debe estar asignado a un grupo. Esto se debe a que un agente de un dispositivo será diferente dependiendo del paciente al que esté asignado, ya que debe disponer de cierta información para ser identificado (nombre, dirección del agente del paciente para enviar datos, etc.). Por esa razón, ambas acciones consultarán primero la existencia o no existencia del dispositivo en la red. Si todo es correcto, añadirán o eliminarán la referencia en el grupo del paciente y procederán igual en el listado de agentes de dispositivos. Un aspecto importante aquí es que los agentes de los dispositivos no se están ejecutando en la misma máquina que el servidor. Esto implica que el gestor no puede iniciar o parar los agentes de forma directa. Por tanto, si se añade un dispositivo nuevo a través de la red, el dispositivo se añadirá, pero se marcará como inactivo. El agente del dispositivo deberá comunicarse con el gestor cuando empiece a captar los datos, y este le enviará los datos necesarios para que pueda empezar a funcionar (como la dirección del agente del paciente al que está relacionado). Esto se detalla en la siguiente sección, que explica las consultas que pueden realizar los agentes al gestor.

Consultas

Los distintos agentes pueden comunicarse sin limitaciones entre ellos si disponen de la dirección adecuada. Esto es un problema en el sistema de este proyecto, que se centra en grupos cerrados de baja extensión basados en los datos de la red social. Los agentes tienen el problema de que no disponen de información sobre el grupo al que pertenecen desde un principio. El gestor sí que dispone de toda esta información, por lo que debe ofrecer métodos para que los demás puedan consultar diferentes aspectos sobre su grupo. Por ejemplo, si un agente recibe un mensaje de otro agente desconocido, podría decidir atender su mensaje de todas formas, o consultar con el gestor para ver si puede confiar en él. Después, deberá decidir si debe almacenar este

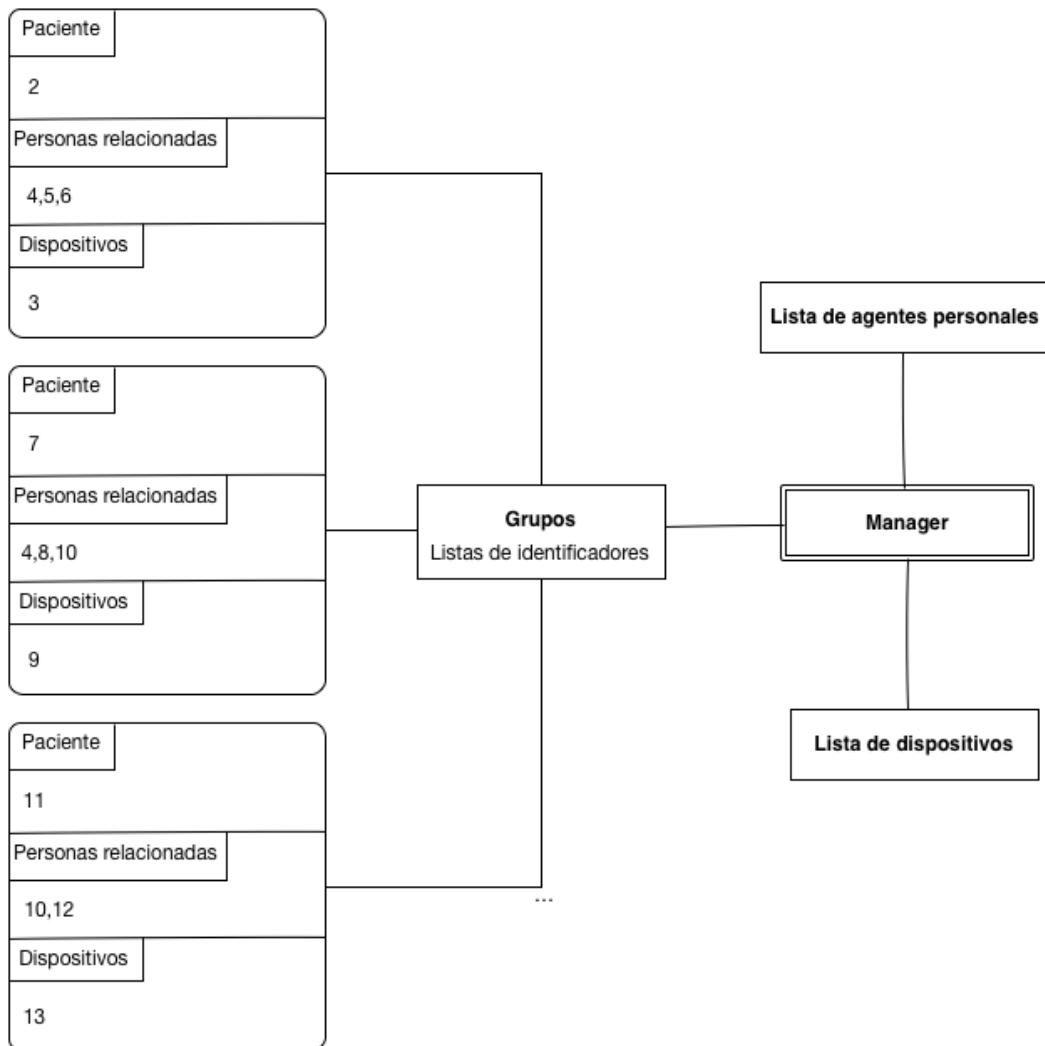


Figura 4.3: Esquema de la arquitectura del gestor de agentes

conocimiento y de qué forma, ya que podría serle de utilidad en un futuro. Existen muchas consultas que puede ser útiles. A continuación se enumeran aquellas que se incluirán en este proyecto:

1. **Obtener todos los agentes del grupo:** puede suceder que un agente quiera enviar un mensaje *broadcast*. Por ejemplo, si un dispositivo ha creado una alerta grave, el paciente puede decidir informar de ello a todas las personas relacionadas.
2. **Obtener agentes dependiendo del rol:** de igual forma que lo anterior, si un agente quiere enviar un mensaje sólo a las personas relacionadas de una

cierta manera. Por ejemplo, puede querer enviar una alerta sólo a los médicos o sólo a sus familiares.

3. **Consultar si un agente es relacionado:** si un agente recibe un mensaje, puede preguntar al gestor si dicho agente pertenece a alguno de sus diferentes grupos. Es decir, si puede confiar en él.
4. **Configurar un nuevo dispositivo:** los agentes de los dispositivos son diferentes al resto. La razón principal es que no se estarán ejecutando en la misma máquina que el gestor. Esto hace que este no tenga poder sobre la instancia del agente. De hecho, al principio sólo dispondrá de un identificador. Es el dispositivo, pues, el que tiene que consultar con el gestor cuando empieza a funcionar. El gestor, ante esa consulta, observará si el dispositivo (que enviará su identificador) se encuentra en la lista de dispositivos disponibles. Si lo está, marcará el dispositivo como activo y responderá al mensaje con los datos necesarios para que el agente del dispositivo pueda comunicarse con su paciente. Después, el dispositivo finalizará su configuración y empezará a generar sus propios datos, que ya podrá enviar al agente del paciente de forma directa.

5

Dispositivo de ejemplo: i-Walker

Toda la arquitectura diseñada quedaría huérfana si no se implementase un dispositivo de ejemplo. El dispositivo escogido para este fin ha sido el caminador robótico i-Walker, desarrollado en la UPC y que forma parte de los proyectos europeos SHARE-it [13] y I-DONT-FALL [29]. En las imágenes de la figura 5.1 se puede observar el caminador visto desde delante y desde detrás, respectivamente.



Figura 5.1: Vista frontal y posterior del caminador i-Walker

El i-Walker[30] es un caminador con tres propósitos principales. El primero y más importante consiste en dar soporte a la movilidad, para que las personas que lo usan puedan caminar de una forma más segura y cómoda. Su segundo objetivo es permitir a estas personas hacer un ejercicio, ofreciendo más o menos resistencia al caminar. Por último, y no menos importante, el caminador ofrece muchos datos sobre los hábitos al caminar del paciente para que puedan ser analizados después.

La funcionalidad del i-Walker está separada en tres áreas diferenciadas:

1. **Análisis:** consiste en obtener información en tiempo real de los sensores y analizarla para permitir: estudiar cómo camina el paciente (velocidad, paradas, etc.), detectar caídas, ver cómo se apoya en el caminador y estudiar cuánta fuerza realiza sobre este.
2. **Soporte:** consiste en dos estrategias: atenuar un porcentaje determinado de las fuerzas necesarias para mover el caminador y forzar al usuario a empujar el caminador en una bajada, haciendo que no tenga que tirar de él. Estos valores pueden ser fijados por su cuidador. Además, no son excluyentes, ya que el paciente puede tener que empujar el caminador en una bajada y a la vez ser tener que realizar menos fuerza en otras situaciones. La segunda estrategia puede usarse también para forzar al usuario a hacer cierto nivel de ejercicio empujando el caminador.
3. **Navegación:** el módulo de navegación recibe comandos de un módulo externo para proveer un soporte de navegación, que ayudará al paciente a alcanzar un destino. Los comandos consisten en gestionar la cantidad de frenado para mantener al caminador en la dirección correcta. Esta función puede ser usada también para que el i-Walker se mueva automáticamente sin ningún usuario, aspecto que puede usarse para que este se mueva de forma autónoma para cargar sus baterías y luego volver a su anterior localización.

Para lograr todos sus funciones, el i-Walker cuenta con diferentes sensores y actuadores. Existen diferentes modelos de caminador, que disponen de algunos tipos diferentes de sensores. No obstante, a continuación se listan todos aquellos que dispone el caminador utilizado y cuáles son sus funciones para lograr los objetivos vistos anteriormente.

- **Raspberry Pi:** el caminador cuenta con un micro computador para procesar todos los datos de los sensores y activar los actuadores. Su función dentro del sistema, por tanto, es vital. Además, también se encarga de guardar datos captados y enviarlos a un servidor para su postprocesado.
- **Motores:** el caminador dispone de cuatro ruedas. Las dos ruedas traseras disponen de dos motores que pueden actuar de forma independiente. Estos motores pueden aplicar tanto aceleración como deceleración, por lo que dependiendo de cómo actúen, pueden proporcionar una ayuda al paciente o un nivel determinado de ejercicio. La Raspberry Pi controlará la velocidad de cada motor dependiendo de los datos captados por los sensores que se describen más adelante y de la configuración introducida por sus cuidadores para cada lado dependiendo de las necesidades del paciente.
- **Acelerómetros:** captan datos de la inclinación del caminador, tanto la incli-

nación lateral (*roll*) como la inclinación vertical (*pitch*). Estos datos pueden ser utilizados para diferentes objetivos. Por una lado, permiten captar el nivel de pendiente, para que el controlador pueda proporcionar ayuda si el paciente está subiendo una cuesta pronunciada, o frenar los motores si está bajando muy rápido. Además, los datos de la pendiente lateral pueden utilizarse para aplicar diferentes velocidades a cada lado del caminador. Por último, estos datos pueden usarse también para detectar posibles caídas, fijando estas cuando se sobrepasa un cierto nivel prefijado de inclinación.

- **Sensores de fuerza:** el caminador monitoriza la fuerza realizada sobre los manillares de este. Estas fuerzas se registran en los tres ejes (*x*, *y*, *z*), por lo que sería posible saber si el paciente se está viendo forzado al caminar. De forma directa, estas fuerzas se usan para detectar cuando el paciente se ha apoyado en el caminador, momento en el cual se activa el control de los actuadores.
- **Frenos:** el paciente puede frenar las ruedas de forma independiente si nota que los motores no ofrecen la ayuda suficiente. Estos frenos son manuales, ya que, en caso de que el controlador decida frenar, utilizará la deceleración que ofrecen los motores. Los frenos tienen dos posiciones diferentes. Por un lado, haciendo fuerza hacia arriba se accionan los frenos con mayor o menor fuerza. Por otra lado, haciendo fuerza hacia abajo se activan los frenos de estacionamiento, que dejan el caminador parado. Para volver a la normalidad, sólo hace falta presionar los frenos de nuevo hacia arriba.
- **LEDS:** indican el estado actual del caminador. Por ejemplo, pueden indicar cuando el ejercicio que se ha fijado ha dado comienzo.

5.1 Datos generados por el i-Walker

El caminador puede generar datos de diferentes maneras. La primera y más básica consiste en la monitorización libre del paciente, que dispondrá del dispositivo y podrá utilizarlo libremente en su entorno. La segunda consiste en una serie de ejercicios predeterminados que el paciente puede realizar.

El caminador generará tres ficheros por cada ejercicio, o irá rellenando uno en caso de que el paciente esté en un entorno libre. Los archivos pertenecientes a cada uno tienen el mismo nombre (UNIX *timestamp* cuando el ejercicio comenzó) con diferentes extensiones. Estos tres ficheros son:

1. **Fichero .usr:** contiene los identificadores del usuario y del caminador. Además, también incluye el UNIX *timestamp* en el que el ejercicio comenzó y los valores

de ayuda y ejercicio (λ y ν) que se fijaron para cada lado del walker.

2. **Fichero .wlk:** contiene los datos de los sensores del caminador, registrados cada cien milisegundos. Cada registro contiene 48 bytes, de entre los cuales 38 bytes son datos de los sensores y 4 bytes corresponden al tiempo en el que se captaron los datos. Para más detalle, en la tabla 5.1 hay una referencia de los datos enviados en cada registro y sus unidades.
3. **Fichero .event:** contiene un pequeño postprocesado de los datos anteriores, mostrando algunos eventos interesantes durante el ejercicio. En la tabla 5.2 hay reflejados todos los eventos calculados.

Tabla 5.1: Referencia de los datos registrados por el i-Walker

Variable	Tipo	Unidad	Rango
Left Hand Force X	Int16-S	cN	[-20000,20000]
Left Hand Force Y	Int16-S	cN	[-20000,20000]
Left Hand Force Z	Int16-S	cN	[-20000,20000]
Right Hand Force X	Int16-S	cN	[-20000,20000]
Right Hand Force Y	Int16-S	cN	[-20000,20000]
Right Hand Force Z	Int16-S	cN	[-20000,20000]
Left Normal Force	Int16-S	cN	[-20000,20000]
Right Normal Force Z	Int16-S	cN	[-20000,20000]
Tilt	Int16-S		
Roll	Int16-S		
Hand Brake Left	Int16-U		0 → free, 1 → braked, 2 → braking
Hand Brake Right	Int16-U		0 → free, 1 → braked, 2 → braking
Estimated Pose X	Int32-S	mm	$[-2 \cdot 10^{31}, 2 \cdot 10^{31} - 1]$
Estimated Pose Y	Int32-S	mm	$[-2 \cdot 10^{31}, 2 \cdot 10^{31} - 1]$
Estimated Pose Orientation	Int16-S	mrad/s	$[0, 2 \cdot 10^{16} - 1]$
Left Speed	Int16-S	cm/s	$[-1000, 1000]$
Left Speed	Int16-S	cm/s	$[-1000, 1000]$

5.2 Pruebas con el caminador

Antes de poder empezar a desarrollar servicios para el caminador en la red social, primero hay que hacer diferentes pruebas reales. Estas pruebas permitirán ver el comportamiento real del dispositivo, fijar límites para los avisos y alertas y, además, apreciar mejor su funcionamiento real.

Tabla 5.2: Referencia de los eventos procesados por el i-Walker

Evento	Parámetro 1	Parámetro 2
Start Walking	Current distance	
Stop Walking	Current distance	Average Relay Force
Continue Walking (alive)	Current distance	Average Relay Force
Angular acceleration warning	Maximum reached	
Linear acceleration warning	Maximum reached	
Low battery	Battery voltage	
Uphill	Tilt angle	Electrical current
Downhill	Tilt angle	
Brake usage	Left brake	Right brake
New user	User Identification	ν and λ

Para poder probar todas las posibilidades que ofrece el i-Walker, se realizaron nueve pruebas diferentes, que cubren todos los aspectos esenciales del caminador. En las siguientes subsecciones se explican cada una de estas pruebas y las conclusiones útiles para el desarrollo.

5.2.1 Medición de ruido con el caminador parado

Los sensores reales, a diferencia de las simulaciones, no son perfectos. Por esa razón, la primera prueba consistió en probar qué nivel de ruido ofrece cada uno de los sensores del caminador. Para ello, se inició una actividad en el i-Walker y se dejó a este sin movimiento y sin presión alguna sobre los manillares.

Los resultados de esta prueba pueden apreciarse en la figura 5.2. En los gráficos se observa que el eje con un mayor ruido es el eje *z*, que puede producir valores de hasta ± 700 cN. A partir de estos datos y los que se obtendrán en las siguientes pruebas, se podrá estimar un valor en el cuál es posible saber cuándo el paciente se ha apoyado en el caminador. Además de los valores de fuerza, el caminar ha marcado un *tilt* de entre 5 y 7 y un *roll* de entre -12 y -13.

5.2.2 Línea recta con paradas

El siguiente paso era comprobar hasta qué nivel medían correctamente los sensores de velocidad. Para comprobarlo, se recorrió un pasillo recto de aproximadamente trece metros en línea recta. El caminador se fijó sin ayuda ni ejercicio. Durante este

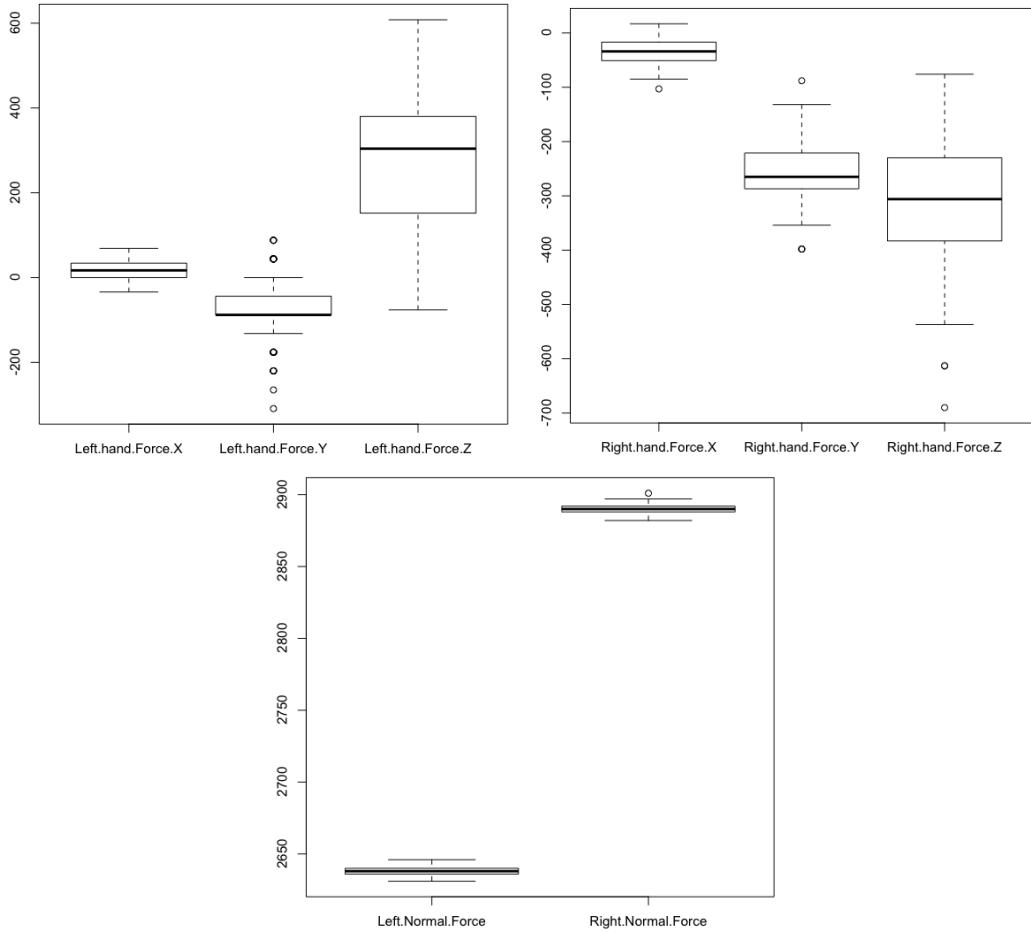


Figura 5.2: Boxplots mostrando los valores de ruido de los sensores de fuerza del caminador: manillares (izquierdo y derecho) y fuerza normal

camino, se fue aminorando la marcha hasta casi parar, pero sin llegar a hacerlo completamente. En ese momento se inició de nuevo la marcha y un poco más adelante se realizó la misma operación, pero está vez sí se paró del todo. Este proceso se realizó un par de veces más para poder confirmar los datos.

En los dos gráficos de la figura 5.3 se pueden observar las velocidades de las ruedas izquierda y derecha durante la prueba. Se aprecia como los sensores sí que detectan velocidades relativamente pequeñas, ya que las falsas paradas que se realizaron muestran una pequeña medida de velocidad. No obstante, hay algo de ruido en la última parada, momento en el cuál se dejó de coger el caminador. Por tanto, se establecerá que el caminador está realmente parado no cuando este marque una velocidad de cero, sino cuando ambas velocidades promedien una media por debajo de 2 cm/s durante medio segundo (cinco ciclos). De esta manera, se podrán eliminar

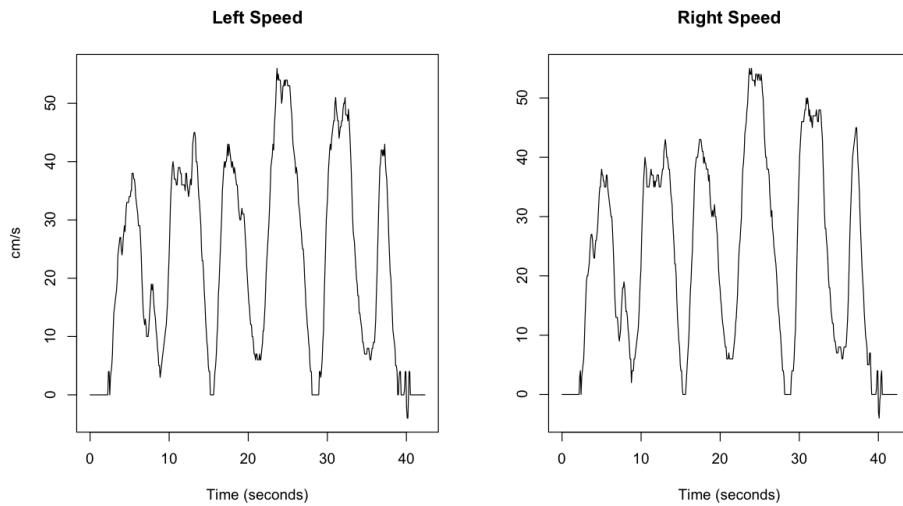


Figura 5.3: Valores de los sensores de velocidad del caminador en línea recta con diversas paradas

aquellos casos en los que el paciente ha parado por alguna dificultad menor y puede seguir después.

5.2.3 Curvas

Se realizó la misma distancia que en la prueba anterior, pero esta vez el caminador realizaba diferentes tipos de curvas con la intención de observar las fuerzas que medía. Se inició la prueba girando primero hacia la derecha y después hacia la izquierda, y repitió este proceso (zigzag). En la cuarta curva (hacia la derecha), se aminoró la marcha sin llegar a parar del todo. Esto también sucedió en la siguiente iteración del zigzag, en la curva hacia la izquierda.

Los resultados de la prueba pueden apreciarse en los gráficos de la figura 5.4. Estos muestran *boxplots* con la media de fuerzas en los manillares en los diferentes ejes. Se puede observar que los ejes *x* e *y* no poseen demasiados valores atípicos, excepto el eje *x*, que posee algunos hasta que el caminador cogió velocidad. Por otro lado, el eje *z* ha producido bastantes valores atípicos hasta que se hubo apoyado en el i-Walker. Estos datos servirán para compararlos con las fuerzas producidas con el caminador configurado con ayuda o ejercicio, que se explican en las siguientes secciones.

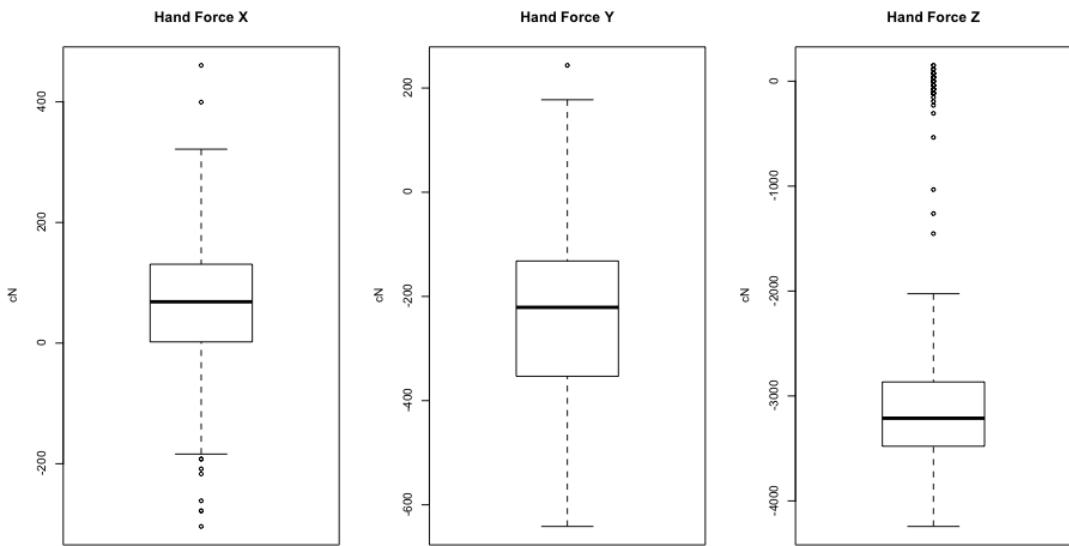


Figura 5.4: Media de fuerzas producidas por el caminador al andar zigzagueando

5.2.4 Curvas con ayuda

En este momento ya se ha observado el comportamiento normal del caminador sin ningún tipo de ayuda o ejercicio. Por tanto, la siguiente prueba consistió en repetir el ejercicio anterior, pero fijando la máxima ayuda que puede ofrecer el i-Walker. El objetivo de este experimento era estudiar las diferencias en las fuerzas que se habían producido en comparación con el ejercicio sin ayudas.

Los resultados (apreciables en la figura 5.5) muestran como el nivel de fuerzas es muy parecido al usar el caminador con ayuda que sin ella. Esto puede deberse a que la prueba se realizó en un terreno completamente llano, y la ayuda del caminador es especialmente útil en pendientes donde el paciente no puede avanzar por sí solo. Además, al realizarse las pruebas por una persona joven y sana, hace que la ayuda proporcionada por el i-Walker no sea demasiado apreciable en este tipo de prueba. Por tanto, no se tendrán en cuenta estos resultados a la hora de implementar los servicios.

5.2.5 Curvas con ejercicio

De nuevo, se realiza el mismo ejercicio que en las dos secciones anteriores, pero esta vez aplicando un nivel de ejercicio máximo y sin ayuda. Otra vez, el objetivo es encontrar la diferencia en el nivel de fuerza de los manillares.

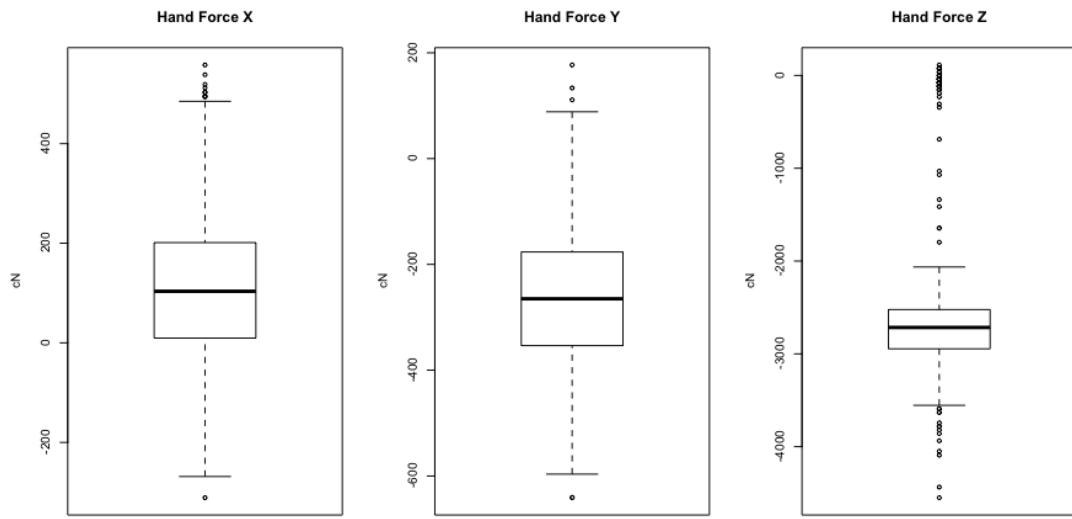


Figura 5.5: Media de fuerzas producidas por el caminador al andar zigzagueando con ayuda (λ) máxima

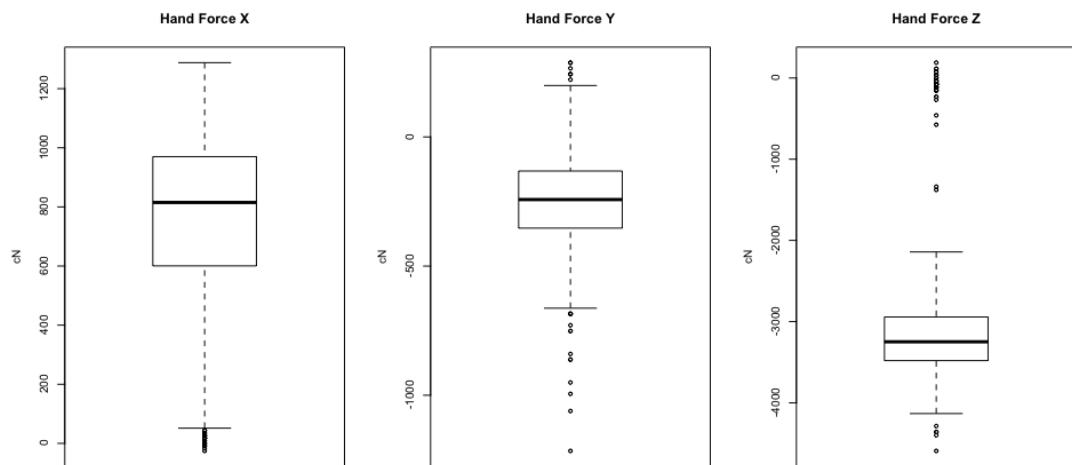


Figura 5.6: Media de fuerzas producidas por el caminador al andar zigzagueando con un valor de ejercicio (ν) máximo

En este caso, los resultados en la figura 5.6 muestran como el nivel de fuerza en los ejes x e y se ha disparado en comparación a las dos pruebas anteriores. Las fuerzas se han multiplicado casi por tres sólo añadiendo un valor de ejercicio al caminador. Estos resultados hacen ver que el nivel medio de fuerza que se ha producido durante el ejercicio puede ser un dato muy a tener en cuenta para analizar cómo se encuentra el paciente. Para ello, se podría mostrar el nivel de ayuda o ejercicio con el que ha sido configurado el caminador y una media de las fuerzas producidas. Es probable

que mostrar todos estos datos es demasiada información. Por esa razón, la mejor idea es mostrar sólo las fuerzas más importantes. En el caso de un caminador, el eje más importante es la fuerza frontal (eje x), es decir, la fuerza que ha hecho el paciente para empujar el caminador. Se ha visto que el eje y también sufre cambios, pero se ha observado que aproximadamente es proporcional al eje x . Por tanto, mostrarlo sería un exceso de información. Por otro lado, el eje z no sería incluido tampoco, ya que, como se observa en los resultados, no sufre grandes cambios dependiendo de cómo se configure el caminador.

5.2.6 Frenos

Se hizo una prueba de detección de frenos, aplicando primero el freno derecho, luego el izquierdo, derecho e izquierdo otra vez. Despues, se aplicaron ambos frenos y, para finalizar, se usaron los frenos de estacionamiento. Primero el derecho, luego el izquierdo (con el derecho aún activo), liberar el derecho y liberar el izquierdo. El objetivo de la prueba era comprobar que los sensores de los frenos funcionaban correctamente y que la frenada se veía reflejada en los valores de velocidad.

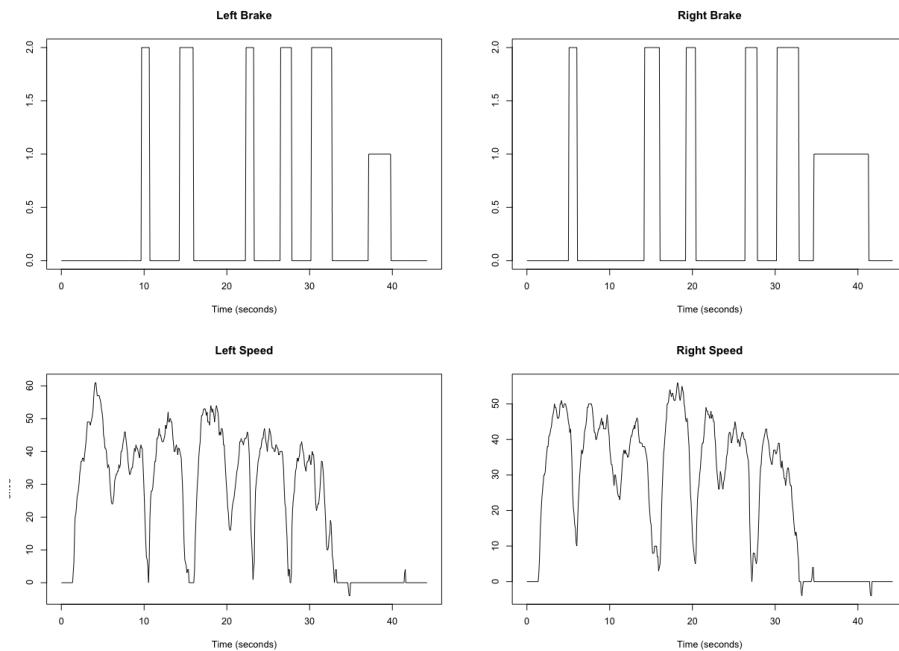


Figura 5.7: Resultados de los sensores en los frenos del caminador y velocidades durante una prueba de frenos

En los gráficos de la figura 5.7 se puede apreciar como todo funciona correctamente. Hay dos factores interesantes en esta prueba. Por un lado, se puede observar que,

durante la primera frenada con ambos frenos, la rueda izquierda marca un velocidad nula, pero no la rueda derecha. Durante ese momento no se pretendía realizar una frenada total. Por otro lado, se vuelve a apreciar un poco de ruido con el caminador parado cuando ambos frenos están activados. por tanto, se confirma que la detección de paradas debe hacerse promediando ambas velocidades.

Esta prueba, además, muestra como se ha hecho un uso extensivo de los frenos de mano. En concreto, se ha estado utilizando alguno de los dos frenos para decelerar el caminador (no estacionado) el 18 % del tiempo. Un número alto en esta variable podría indicar que el paciente no se siente cómodo o seguro al caminar. Más en concreto, los servicios de ejemplo podrían generar un pequeño aviso en el resumen de la sesión indicando este aspecto si se supera el límite del 10 % de uso de los frenos durante los ejercicios.

5.2.7 Acelerómetros

Uno de los objetivos de los servicios del caminador es la detección de posibles caídas. Es posible detectarlas observando los valores de inclinación y fijando un límite a estos valores, a partir de los cuales se puede considerar que el paciente ha caído. Durante la prueba, primero se inclinó el caminador hacia la derecha, en lo que podría no ser una caída. Después, se inclinó más para ver qué valor podría ser el límite. Más tarde se repitió este proceso, pero volviendo a colocar el i-Walker en su posición original para apreciar mejor los valores de los sensores y realizando más ángulos. Los pasos fueron: ángulo que podría no ser una caída, ángulo que representaría una caída y por último una caída extrema (caminador en el suelo). Se realizaron también pruebas similares hacia la izquierda y también hacia atrás (pendiente), para confirmar que los límites fueran los mismos.

Una vez obtenidos los datos (gráficos en la figura 5.8), se pueden fijar los límites de una caída en aproximadamente ± 400 en ambas variables. No obstante, se fijará un valor de ± 800 para confirmar que la caída se ha producido realmente. Se ha fijado de esta manera debido a que, si la caída es real, el caminador registrará estos datos extremos en algún momento durante la caída.

5.2.8 Fuerzas

El caminador mide las fuerzas de los manillares en los tres ejes. Para identificar qué eje representa cada una de estas fuerzas en el mundo real y qué valores podría alcanzar, se realizaron diferentes fuerzas en diferentes direcciones e intensidades.

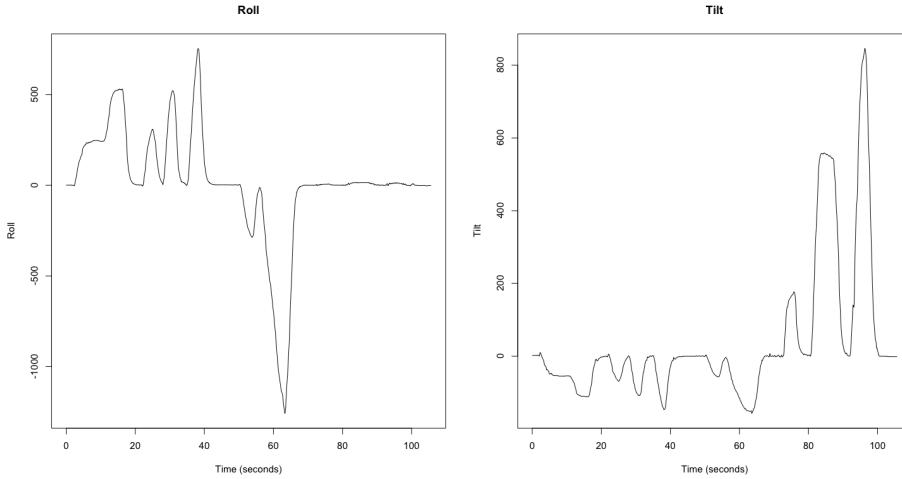


Figura 5.8: Valores obtenidos de los acelerómetros del caminador en una prueba de inclinaciones

Primero se hizo fuerza hacia abajo, hacia la derecha y después hacia la izquierda. Se repitió este proceso con más fuerza dos veces más. Finalmente se hizo otra prueba de fuerza hacia abajo. El objetivo de esta prueba era, además de identificar los ejes, fijar unos límites a partir de los cuales se generaría un aviso por exceso de fuerza o para detectar cuando el paciente se ha apoyado en el caminador.

Los resultados (gráficos de la figura 5.9) muestran que la fuerza vertical se representa en el eje z , siendo los valores negativos la fuerza hacia abajo y positivos hacia arriba; el eje y representa las fuerzas laterales, negativo hacia la derecha y positivos hacia la izquierda; y el eje x las fuerzas hacia delante y hacia atrás.

También se puede fijar la fuerza en la que se establece si el paciente se ha apoyado en el caminador. Para eso hay que mirar el eje z negativo. Esta fuerza se ha fijado cuando se baja del límite de -1000 cN durante medio segundo. Por otro lado, se producirá un aviso por el nivel de fuerza si se superan los ± 8000 cN en el eje z o los ± 3000 cN en el eje y . En el eje x no se han fijado unos límites, ya que el paciente podría estar empujando el caminador hacia delante con una cantidad de fuerza indeterminada en algunas ocasiones sin que existiera ningún problema.

5.2.9 Prueba real

En este punto ya se habían obtenido todos los datos deseados de los diferentes sensores del caminador. No obstante, se decidió hacer una prueba larga en terreno real, con diferentes terrenos e inclinaciones. Para ello, se paseó con el caminador por

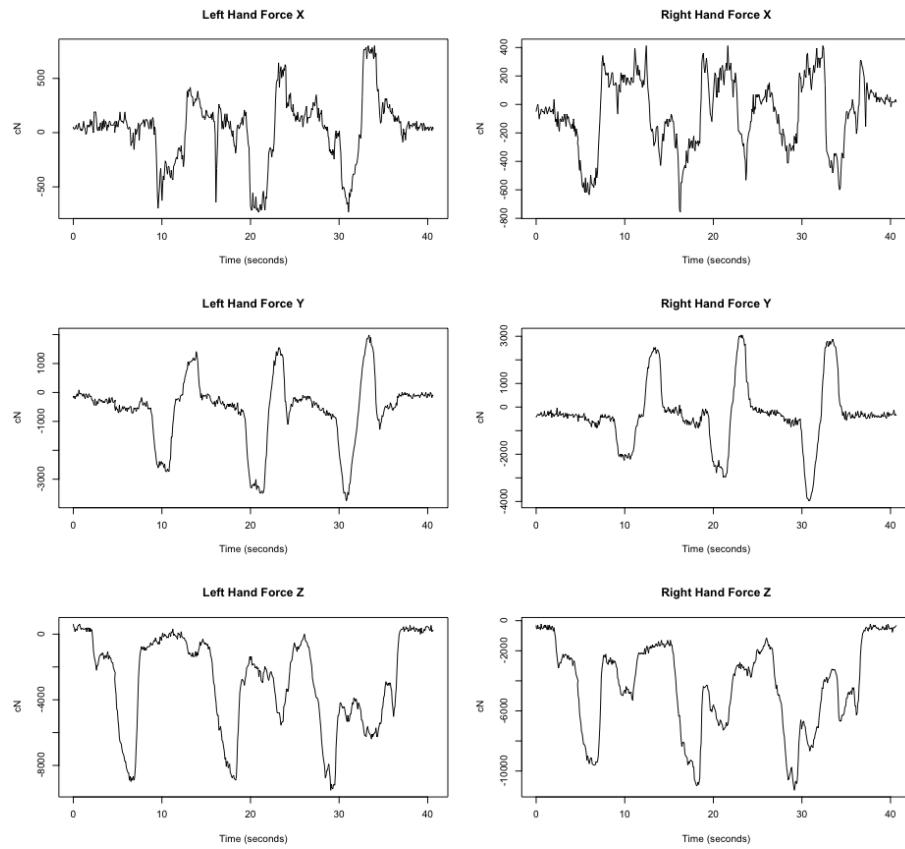


Figura 5.9: Fuerzas (x , y , z) captadas por el i-Walker durante una prueba de fuerzas

el campus con un nivel de ayuda medio. Primero, se atravesó la plaza de la biblioteca, produciendo vibraciones en el caminador. Después, se bajó por una pendiente para ver cómo el caminador frenaba debido a la ayuda configurada. Para finalizar, se atravesó una recta sin bacheado y después se subió una pendiente, donde el caminador encendió los motores para ayudar a subirla.

La prueba no tenía ningún objetivo en particular, sino que se pretendía captar una mayor cantidad de datos para poder hacer pruebas con ellos todo lo aprendido en las anteriores pruebas. Por esta razón, no se incluye ningún tipo de gráfica de las fuerzas o velocidades captadas durante el recorrido.

5.3 Procesado de datos

A través de estos experimentos pueden fijarse diferentes aspectos a tener en cuenta en el procesado de cada ejercicio. El objetivo del agente del paciente es informar

sobre el comportamiento del paciente, alertar sobre posibles peligros e intentar que mejore su estado de ánimo. Todo esto es posible mediante el análisis de los datos generados por los diferentes sensores del caminador. Cada uno de estos aspectos puede conseguirse de las siguientes maneras:

1. El paciente estará informado de su actividad con el caminador mediante la generación de informes sobre un ejercicio o sesión (grupo de ejercicios durante un periodo corto de tiempo). Estos informes contendrán información de la distancia recorrida, el uso de frenos, nivel de ayuda y ejercicio, porcentaje de subidas y bajas, un listado de eventos sucedidos durante el ejercicio y un análisis sobre la tendencia en el tiempo.
2. Los diferentes agentes serán alertados si se producen valores extremos en alguno de los sensores. De forma específica para el dispositivo, estos valores incluirán inclinaciones extremas (caídas o posibles caídas) y valores de fuerza superiores a los límites estudiados.
3. Mejora del estado de ánimo del paciente: el agente del paciente asignado al caminador guardará una estimación de su estado de ánimo. Puede determinarse que un aumento en la distancia caminada durante el tiempo puede implicar también una mejora de su estado general (incluyendo el estado de ánimo). Por tanto, este valor puede ser usado para intentar mejorar su ánimo haciendo que camine más. Las modificaciones de esta estimación serán posibles prediciendo la tendencia de la distancia que se va a realizar en un futuro usando los datos pasados, y cambiando el valor dependiendo de esta predicción. Esto se puede realizar usando una regresión lineal, aspecto que se explica en más detalle en la sección 9.3.4. Además, esta estimación se utilizará para alertar a los demás agentes relacionados si se estima que es muy baja.

6

Implementación de la red social

Una vez acabado el diseño es posible empezar a implementar todos los sistemas. Como ya se ha visto anteriormente, el desarrollo está dividido en tres grandes categorías: la red social, el sistema multi-agente y la interfaz entre estos dos sistemas. Durante este capítulo se explicará como se ha llevado a cabo la implementación del primero de estos sistemas, cómo se ha estructurado el código y los mayores problemas que han surgido. Los dos siguientes capítulos harán lo respectivo para el MAS y la interfaz.

6.1 Motor para la red social

Debido a que una red social requiere una cantidad de código demasiado grande, no es posible desarrollarla desde cero implementando sólo aquellos aspectos importantes para el proyecto. Por tanto, el paso más importante a la hora de empezar la implementación es escoger qué motor se va a utilizar, intentando que este sea lo más adecuado para el desarrollo.

Varios aspectos son importantes a la hora de escoger el motor. En primer lugar, se busca uno que sea estrictamente de **código libre** (licencia GPL [25], MIT [33], Apache [4], etc.). De esta manera, se podrán realizar todas aquellas modificaciones que se deseen, que podrían modificar el núcleo de la red social de forma notable. Además, de esta forma sería posible mantener la licencia libre para el resto del proyecto. En segundo lugar, el motor debe ofrecer un balanceo de sus características, no demasiado simples o demasiado complejas. Cualquiera de estos extremos haría que aumentara el trabajo a realizar para implementar las funcionalidades faltantes o, en su defecto, eliminar las sobrantes (tarea que podía llegar a ser difícil si estas se encuentran en el núcleo del motor). Por último, el motor debe ofrecer un gran **dinamismo** a la hora de desarrollar nuevas modificaciones. Esto se debe a que durante el proyecto se modificarán algunas partes muy básicas de una red social, por lo que se tiene que estar seguro de que será posible realizar todas estas modificaciones antes de que empiece el desarrollo.

Fijadas las anteriores exigencias, se inició la búsqueda del motor adecuado. Se barajaron diferentes tecnologías y lenguajes de programación. Entre las opciones

descartadas se encontraban Community Engine [11] (implementado en Ruby¹), o Drupal [18]. Al final, las dos opciones a escoger fueron BuddyPress [52] y Elgg [14].

Elgg y BuddyPress ofrecen diferentes ventajas y a la vez desventajas respecto a la otra. Por un lado, Elgg está desarrollado desde cero para ser un motor de redes sociales, mientras que BuddyPress es un añadido encima de Wordpress, un gestor de contenidos. Debido a esto, por otro lado, la comunidad detrás de BuddyPress es mucho mayor que la que se puede encontrar en Elgg. Sin embargo, ambos ofrecen la mayoría de cualidades que se buscan para la red social, por lo que más que comunidad, se prefiere que el desarrollo sea fácil y dinámico.

Además de las características básicas, ambos motores disponen de algunos atributos propios. Elgg está pensado para que sea fácil para el desarrollador realizar modificaciones, incluso a bajo nivel. Además, incluye una API con la cuál es posible exponer funciones hacia el exterior de la red, con lo que el desarrollo de la interfaz podría ser más rápido. Por otro lado, BuddyPress ofrece también un sistema maduro de desarrollo, que lleva usándose con Wordpress desde hace muchos años. Además, permite también una mayor usabilidad y flexibilidad para administrar el sitio.

Ambos motores, por tanto, ofrecen muy buenas características. Elgg tenía la principal ventaja de ofrecer servicios Web por defecto, pero la mayor comunidad de BuddyPress era un buen aliciente. Para acabar de decidir, finalmente se decidió realizar una prueba real de desarrollo. En esta prueba, se implementó un pequeño añadido a la red en ambos motores, con el objetivo de comprobar en cuál de los dos motores resultaba más fácil y cómodo realizarlo. La prueba finalizó con Elgg como ganador. Elgg demostró ser más dinámico a la hora de realizar modificaciones al motor, mientras que BuddyPress, aún siendo más usable, ofrecía una mayor dificultad en determinados aspectos del desarrollo. En resumen, **Elgg será el motor escogido para implementar la red social**.

6.2 Elgg

Una vez escogido el motor de redes sociales en el que se va a implementar el sistema, el primer paso es estudiarlo a fondo para optimizar el código a la forma de trabajar de sus desarrolladores, permitiendo así un código más óptimo y un trabajo menos laborioso.

¹Lenguaje de programación dinámico y multi-paradigma

6.2.1 Sistema

Elgg [14] es un motor basado en el lenguaje de programación PHP [37]. Esto significa que existirá un servidor Web que, dependiendo de las consultas, ejecutará diferentes algoritmos dentro de la red social, produciendo finalmente código HTML que será interpretado por el navegador del usuario.

Para funcionar correctamente, Elgg necesita del servidor Web Apache [2], con el módulo `mod_rewrite` activo. Además, una versión de PHP igual o superior a 5.2 es necesaria para que Elgg funcione sin errores. Por otro lado, para guardar todos los datos de la red, Elgg hace uso de la base de datos MySQL [34], la versión de la cual debe ser igual o superior a 5. Por tanto, para mantener la filosofía libre del proyecto, se recomienda hacer uso de un entorno Linux (LAMP).

6.2.2 Modelo de datos

Preguntarse cómo se va implementar el grafo de la red social es el primer paso que hay que realizar para empezar el desarrollo por el buen camino. Elgg ofrece ya un modelo de datos [15] maduro y muy dinámico, que permitirá implementar todo lo que se ha visto en el diseño de la red **sin modificar el núcleo del motor**. De este modo, el mantenimiento del código a través del tiempo será más eficiente. En la figura 6.1 se muestra un esquema básico de este modelo. A continuación se explicará en más detalle cada uno de sus elementos.

Entidades

El elemento básico del modelo de datos se conoce como entidad, `ElggEntity` en la implementación. Cada una de estas entidades representarán un vértice en el grafo G de la red, aunque algunas de estas entidades no estarán incluidas en el grafo debido a que son usadas por Elgg para su funcionamiento interno.

Toda entidad de la red dispondrá de ciertos datos básicos para regular su lectura y escritura. El más importante de estos atributos es el *Globally Unique Identification* o `guid`. Este identificador es único entre todas las entidades, y permite, por tanto, identificarlas de forma única. Otros atributos importantes para la implementación son `owner_guid`, que indica el usuario dueño de la entidad, y `subtype`, una cadena de caracteres que determina el tipo de entidad.

Aumentando otro nivel de abstracción, Elgg proporciona cuatro tipos de entidades principales:

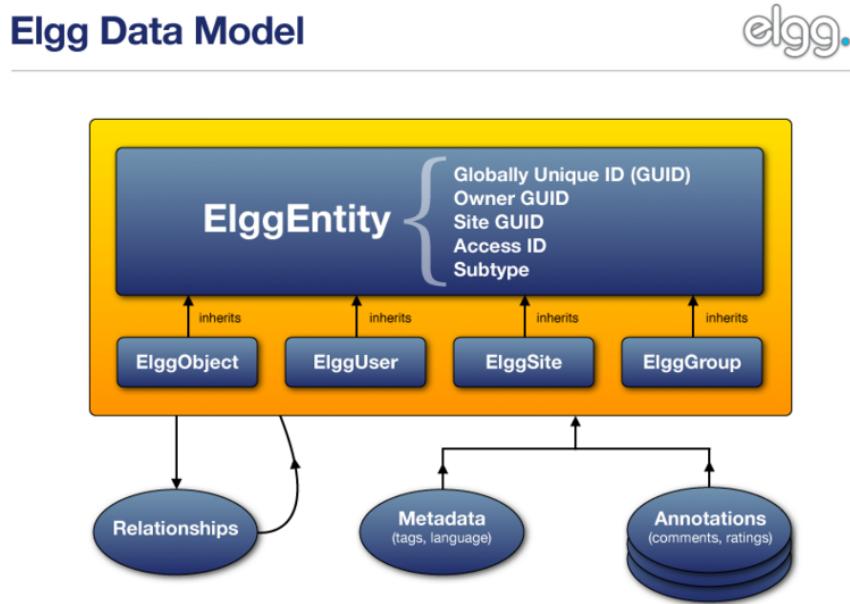


Figura 6.1: Modelo de datos de Elgg

1. **ElggSite**: representa un sitio completo. No es importante para esta implementación, ya que se dispondrá de un único sitio. Además, este tipo de entidad no representa a ninguno de los vértices descritos durante el diseño de la red. Por tanto, no se explicará en más detalle.
2. **ElggUser**: representa a todos los usuarios humanos de la red. Además de los atributos heredados de **ElggEntity**, se pueden encontrar otros como: el nombre a mostrar en la red (`name`), un nombre de usuario único (`username`), la contraseña (`password`) y la dirección de correo electrónico (`email`). Además de estos atributos, cada usuario dispone de algunos datos de control más, aunque no necesarios para la implementación. Por esta razón, sólo se enumerarán. Estos son: `salt`, `language`, `code`, `last_action`, `prev_last_action`, `last_login`, `prev_last_login`.
3. **ElggObject**: representa cualquier otro vértice del grafo G que no sea una persona. Debido a esta generalidad, sólo dispone de dos nuevos atributos además de los heredados de **ElggEntity**: un nombre (`name`) y una descripción (`description`). Por tanto, se necesitará algo más para poder representar de forma completa todo el diseño.
4. **ElggGroup**: representa un grupo de entidades. Al igual que **ElggSite**, no es importante para esta implementación y, de hecho, no existirá ninguna instancia de este tipo de entidad.

Metadata

El modelo de entidades permite representar cualquier objeto no específico. No obstante, muchas veces es necesario incluir más datos en la entidad para desarrollar el diseño que se ha realizado. Elgg ofrece una forma de extender las entidades añadiendo metadatos a estas, de tal forma que una entidad puede tener un número indefinido de atributos personalizados.

Hay que tener en cuenta que los metadatos son entidades propias (de la clase `ElggMetadata`) que heredan de `ElggEntity`. Esto significa que tienen dueño y valores asignados para los permisos. Por defecto, el dueño de los metadatos se asigna al usuario identificado en ese momento, mientras que los permisos se heredan de la entidad a la que están adheridos. Además, estos valores cambian cada vez que se modifican los metadatos. Por estas razones, hay que tener especial cuidado al crear o modificar metadatos, ya que los sistemas o los usuarios privilegiados de la implementación crearán y modificarán datos de otros usuarios.

Anotaciones

Existen ciertos datos que están siempre ligados a cierta entidad de forma fija. Por esta razón, en Elgg podemos encontrar las anotaciones. Las anotaciones son pequeñas porciones de datos que acompañan a una entidad. Por ejemplo, un comentario en una actividad de un paciente se consideraría una anotación a esa actividad. Elgg las guarda utilizando la clase `ElggAnnotation`. Para añadir una anotación a una entidad, necesitamos saber el nombre de la anotación (por ejemplo, comentario), la entidad a la que irá adjunto, el contenido de la anotación y algunos atributos de control. Debido a que no forma parte fundamental de esta implementación, no se verán en más detalle.

Relaciones

En este punto, se ha visto como implementar en Elgg los vértices de la red. El siguiente paso es conectar estos vértices con diferentes tipos de relaciones. De esta forma se consigue ya la estructura completa de la red y ya sería posible implementar todo el diseño. Para ello, Elgg dispone de un modelo simple y potente. Este modelo permite relacionar cualquier par de entidades de forma personalizada, es decir, dos entidades pueden estar conectadas entre ellas de un número indeterminado de formas. Para conseguir esto, Elgg necesita los números de identificación únicos (`guid`) de las dos entidades, y también una cadena de caracteres con el nombre de la relación. En

el código 6.1 se puede observar cómo añadir, comprobar o eliminar relaciones entre dos entidades.

Hay que tener en cuenta que estas relaciones no son recíprocas, es decir, si se quiere conseguir una relación bidireccional habrá que añadir dos relaciones del mismo tipo en Elgg cambiando el orden de las entidades. Esto es muy importante, ya que puede llevar a errores de programación importantes.

Código 6.1: Ejemplos de operaciones con relaciones entre entidades (Elgg)

```

1  <?php
2  $patient = get_user_by_username('patient');
3  $doctor = get_user_by_username('doctor');
4  // Agregar la relacion entre doctor y paciente
5  add_entity_relationship(
6      $patient->guid,
7      'is_patient',
8      $doctor->guid
9  );
10 // Comprobar la relacion
11 $is_patient = check_entity_relationship(
12     $patient->guid,
13     'is_patient',
14     $doctor->guid
15 );
16 // Eliminar la relacion
17 remove_entity_relationship(
18     $patient->guid,
19     'is_patient',
20     $doctor->guid
21 );
22 ?>

```

Observaciones

Una vez visto el modelo de datos de Elgg, podemos decir que, aunque el modelo de datos es realmente dinámico, quizás no sería el motor más adecuado si existiese un gran número de usuarios. En ese caso se necesitaría un modelo de datos más estático que pudiera ser personalizado a bajo nivel para las necesidades específicas proyecto. Esto implicaría diseñar una red social desde cero. No obstante, hay que destacar

que este aspecto no forma parte de este proyecto, pero sí podría ser algo a tener en cuenta en otros proyectos futuros.

6.2.3 Desarrollo de añadidos y modificaciones

Como ya se ha comentado anteriormente, Elgg permite un gran dinamismo en el sistema sin necesidad de modificar el código del núcleo. Este dinamismo, sin embargo, se paga **siguiendo estrictamente** la estructura esperada por Elgg a la hora de programar los *plugins* [16].

La raíz de Elgg cuenta con diferentes ficheros y directorios. El único directorio que debe modificarse de todos ellos es `mod`, del inglés *modifications*. Cualquier cambio fuera de este **podría perderse** en una futura actualización de la versión del motor, por lo que no se recomienda bajo ningún concepto.

Dentro del directorio de modificaciones existen tantas carpetas como *plugins* presentes. Es decir, cada añadido o modificación debe tener su propia carpeta, no puede estar incluído dentro de la carpeta de otro (por ejemplo, si se están añadiendo características a otro *plugin*). El nombre de este directorio es importante, ya que se utilizará después para referenciar archivos importantes.

De nuevo, dentro del directorio de cada *plugin* la estructura debe seguir unos patrones. Como mínimo, un *plugin* debe disponer de dos ficheros fundamentales: `manifest.xml`, que describe sus características y dependencias, y `start.php`, el código a ejecutar cuando este está activo.

Para las modificaciones más complejas, Elgg hace uso de una arquitectura MVC². Este patrón separa el procesado de información en tres niveles diferentes: modelo, vista y controlador. En primer lugar, el modelo gestiona los datos persistentes en el sistema (usuarios, informes, notificaciones...). El controlador es el encargado de pedir estos datos al modelo, o modificarlos, dependiendo de las peticiones de los usuarios. A través de estos datos, el controlador puede llamar a diferentes vistas, que, a través de los datos del modelo y los generados por el propio controlador, generarán lo que el usuario final verá en su pantalla.

Por tanto, la jerarquía de directorios de cada *plugin* obedece en mayor o menor medida al patrón MVC. A continuación se especifican todas las diferentes carpetas que pueden existir y cómo funcionan.

- **lib:** contiene bibliotecas de ayuda para el resto de archivos. En caso de que el añadido proporcione un nuevo tipo de entidad o modifique una ya existente,

²Patrón de diseño basado en modelos, vistas y controladores

este directorio es el adecuado para colocar los modelos del patrón MVC. Para aumentar el rendimiento, las bibliotecas no se registran o cargan de forma automática, sino que se debe hacer de forma manual. Para registrarlas, se usa la función `elgg_register_library`, que puede llamarse en el fichero `start.php` del *plugin*. A partir de este momento, la biblioteca puede ser cargada por cualquiera usando la función `elgg_load_library`. El único requisito es que debe ser registrada antes. Por tanto, si se usa esa biblioteca en otro *plugin*, se deben incluir las dependencias necesarias en el archivo `manifest.xml`.

- **views:** contiene los archivos necesarios para generar el contenido que visualizarán los usuarios. De nuevo, se sigue una jerarquía muy estricta. El directorio contiene diferentes niveles, que indicarán qué tipo de contenido se está generando. El primero de estos niveles indica el tipo de salida. El tipo más usado es `default`, que generará código HTML o CSS. Sin embargo, también es posible utilizar cualquier otro tipo, como JSON o RSS. El segundo nivel indica el módulo al que pertenece la vista. La mayoría de las ocasiones será el propio nombre del *plugin*, pero también podría ser el nombre de otro. En este último caso, la nueva vista se añadiría a las ya existentes en el otro *plugin* o se reemplazaría. Existen algunos módulos especiales en este nivel. Por un lado nos encontramos `css` y por otro lado `forms`. Estos directorios indican a Elgg dónde puede encontrar los estilos y los formularios. Por último, el siguiente nivel contiene los ficheros de las vistas o, en el caso de los módulos especiales, de nuevo el nombre del módulo al que pertenecen (se añade otro nivel).
- **pages:** contienen un tipo de controlador en el patrón MVC. Son los encargados de generar las páginas que verán los usuarios, haciendo llamadas a los modelos y generando una disposición del espacio, que llenarán con diferentes vistas, que podrán ser llamadas con datos generados en el propio controlador o obtenidos de los modelos. La jerarquía de carpetas sigue el mismo patrón que en el punto anterior, pero sin existir módulos especiales.
- **actions:** contienen otro tipo de controladores. Se utilizan para procesar datos introducidos por los usuarios en los formularios. La gran diferencia con `pages` es que, en este caso, el objetivo principal es guardar datos en el modelo y no obtenerlos. Por otro lado, al igual que las bibliotecas, las acciones tienen que ser registradas manualmente en el archivo `start.php`.
- **languages:** ayudan a la internacionalización de la red. Contienen traducciones a diferentes idiomas del contenido a mostrar, para que este no se vea limitado a un único lenguaje. La jerarquía es la misma que en los anteriores directorios, por lo que es posible añadir o modificar traducciones de otros *plugins* sin modificarlos directamente.

Otro aspecto importante a la hora de desarrollar una modificación son los *menu items*. Elgg permite modificar los elementos de los menús de forma dinámica, sin tener que modificar el código original. En concreto, se pueden añadir elementos a un menú usando la función `elgg_register_menu_item` y eliminarlos con la función `elgg_unregister_menu_item`. De esta manera, sería posible mostrar un menú u otro dependiendo del rol que desempeñe un usuario.

Por último, durante la ejecución del código en el núcleo o en un *plugin* se pueden producir eventos, que pueden ser interceptados por otros *plugins*. Elgg establece dos tipos de eventos: `events` y `plugin hooks`. Los `events` se producen cuando Elgg está siendo cargado, o cuando algo ha sido creado, actualizado o eliminado. Por otro lado, los `plugin hook` son más específicos, y se lanzan cuando se está produciendo algún tipo de acción y alguna de sus partes pueden ser sobreescritas por otros *plugins*. En ambos casos, es posible interceptar los eventos e introducir código propio en ellos. Por tanto, añade la posibilidad de modificar determinadas partes del núcleo de Elgg o de otros mods sin necesidad de una modificación directa.

6.3 Arquitectura de los *plugins*

Una vez se sabe cómo funciona el sistema de modificaciones de Elgg, hay que fijar el número de *plugins* que se usarán, cuál será el objetivo de cada uno y qué dependencias habrá entre ellos. Con el diseño de la red delante y los *plugins* ya desarrollados por la comunidad, se decidió desarrollar un total de **diez modificaciones propias**, que harán uso de dos implementadas por la comunidad, además del núcleo de Elgg.

El esquema de la figura 6.2 muestra la arquitectura escogida para implementar la red social. El *plugin rs* no tiene otra utilidad más que agrupar a todos los demás, ya que en Elgg las modificaciones tienen que activarse manualmente una a una. Además de esto, puede apreciarse como el núcleo de las dependencias apuntan hacia el gestor de usuarios por una parte, y al gestor de la API que comunicará con el MAS. En las siguientes secciones se explica en más detalle el objetivo y comportamiento de cada uno de estas modificaciones, excepto el de la interfaz de la red, que se detalla en el capítulo 8.

6.4 Usuarios, roles y relaciones

En el diseño del grafo G de la red social, la conexión entre los diferentes usuarios es el punto más importante. La implementación de este aspecto puede diferenciarse en

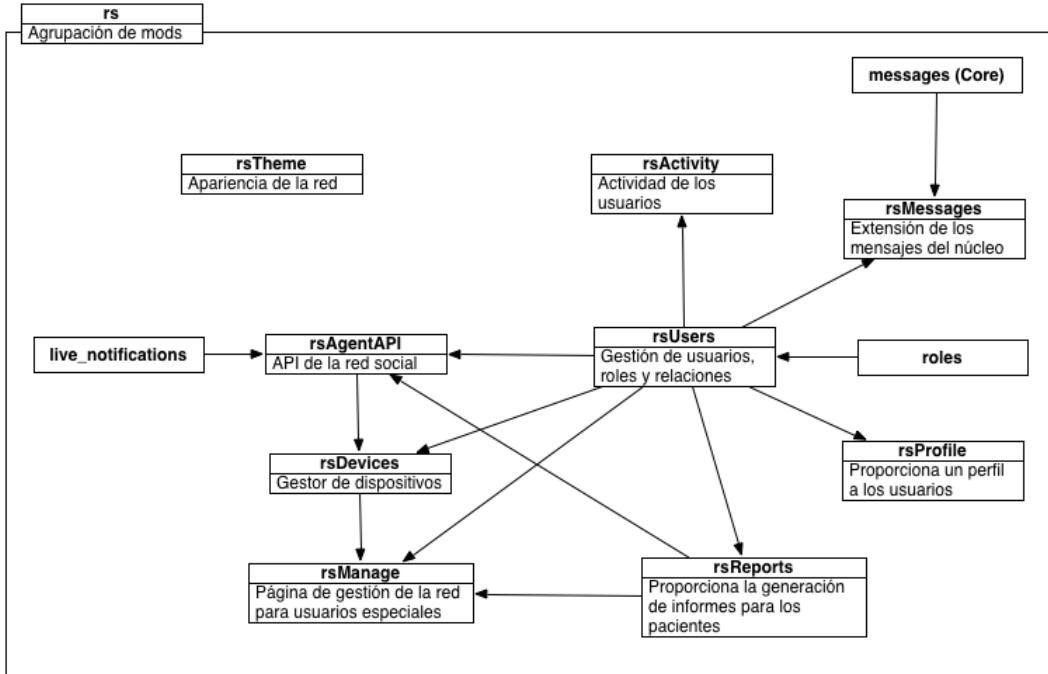


Figura 6.2: Arquitectura de las modificaciones realizadas Elgg para el proyecto

dos etapas: primero, implementar los diferentes roles que puede tener cada usuario; y segundo: establecer las diferentes conexiones entre los usuarios. La implementación de todas estas características se ha llevado a cabo en el *plugin rsUsers*, el más importante de la red social y del que beben la mayoría de los demás.

6.4.1 Roles

La primera parte del desarrollo se ha realizado utilizando un *plugin* de la comunidad Elgg, que ofrece un framework sencillo para dotar de roles a los usuarios. En el interior de este, los roles están implementados de la siguiente manera: por cada rol definido en el archivo de configuración se crea un `ElggObject` con subtipo `role`. Esta entidad dispone de ciertos metadatos para regular sus permisos (acceso a páginas, acciones, vistas, etc.). Para dotar a un usuario de cierto rol, es necesario añadir una relación `has_role` entre la entidad del usuario y la entidad del rol. Para entender mejor su funcionamiento, el código 6.2 incluye algunos ejemplos sobre cómo añadir, comprobar y eliminar roles a un usuario.

Para poder incluir nuevos roles, el *plugin* incluye un fichero que añadirlos a través de código. Para no modificarlo directamente, también ofrece un *hook*, que puede

utilizarse para añadir roles desde `rsUsers`. Para que funcione correctamente, se tiene que ejecutar antes de la modificación propia. Esto se consigue mediante las dependencias definidas en el fichero `manifest.xml`.

Código 6.2: Ejemplo de operaciones con roles (Elgg)

```
1 <?php
2 $patient = get_user_by_username('patient');
3 // Obtener la entidad del rol paciente
4 $role_patient = roles_get_role_by_name('PATIENT_ROLE');
5 // Fijar el rol del paciente
6 roles_set_role($role_patient, $patient);
7 // Obtener el rol del paciente
8 $role = roles_get_role($patient->guid);
9 ?>
```

Ahora sólo resta añadir los roles correspondientes. Como se vio en el diseño de la red, en este proyecto se incluirán sólo algunos roles básicos, que podrían extenderse en un futuro. Estos roles incluyen a los pacientes, familiares, doctores, enfermeros, secretarios y administradores. Los permisos para cada grupo se fijan en dos sitios diferentes. En primer lugar, el archivo de configuración de roles permite realizar una configuración básica de estos, para limitar el acceso a ciertas página o la realización de algunas acciones. De forma más concreta, cada controlador es responsable de los usuarios que pueden interactuar con él.

6.4.2 Relaciones

Por defecto, Elgg solo ofrece la relación de amigos entre los usuarios. Es necesario, por tanto, desarrollar una solución propia para el proyecto. En primer lugar, antes de añadir nada, hay que eliminar todo aquello relacionado con la relación de amigos del sistema. Después, hay que implementar los modelos, vistas y controladores para gestionar las nuevas relaciones.

Para los roles definidos en este proyecto, se usarán las siguientes relaciones dependiendo del rol de los usuarios implicados:

1. `is_patient` e `is_doctor` serán las relaciones entre paciente y doctor.
2. `has_relative` e `is_relative` se incluirán entre los pacientes y sus familiares.
3. `has_nurse` e `is_nurse` se llevarán a cabo entre los pacientes y sus enfermeros.

4. `is_related`: relación general entre usuarios. Este tipo de relación se incluirá junto a cada una de las relaciones anteriores. Se ha decidido incluirla para rebajar el nivel de complejidad de algunas consultas si sólo existieran relaciones específicas. Esto se debe a que este tipo de relación es suficiente para la mayoría de consultas.

Una vez fijadas las relaciones, se puede proceder a implementar el modelo que las gestione. En el código 6.3 pueden apreciarse las diferentes funciones que permiten trabajar con estas relaciones. Con ellas se pueden consultar las relaciones de un usuario, si está relacionado con otro, si puede tener o no relaciones (dependiendo de su rol) o si el usuario tiene permisos para modificar relaciones o dispositivos de otro usuario. Además de consultar, también permite añadir o eliminar la relación entre dos usuarios. Estas nuevas relaciones tienen que obedecer a las reglas que se han fijado en la sección anterior. En caso contrario, si la relación no es plausible, no se añadirá ningún tipo de relación (ni la relación `is_related` ni la relación específica).

Código 6.3: Modelo para la gestión de relaciones en la red social

```

1  <?php
2  // Obtener usuarios relacionados
3  relations_get_related_users($user_guid);
4  // Obtener aquellos usuarios relacionados que son pacientes
5  relations_get_patients($user_guid);
6  // Comprobar si dos usuarios estan relacionados
7  relations_are_related($user1_guid, $user2_guid);
8  // Comprobar si un usuario puede tener relaciones (rol)
9  relations_can_have_relations($user_guid)
10 // Comprobar si el usuario actual esta relacionado con otro
11 relations_related_to($user_guid);
12 // Comprobar si el usuario actual tiene permisos sobre otro
13 relations_can_modify_relations($user_guid);
14 relations_can_modify_devices($user_guid);

15
16 // Agregar o eliminar relacion entre dos usuarios
17 relations_add_relation($user1_guid, $user2_guid);
18 relations_delete_relation($user1_guid, $user2_guid);
19 ?>

```

6.4.3 Páginas

Los diferentes roles tienen que tener la posibilidad de gestionar usuarios y relaciones. Diferentes páginas están disponibles dependiente de los roles de los usuarios. A continuación se enumeran todas ellas:

1. **Usuarios:** todas las páginas para modificar usuarios necesitan permisos de gestión, por lo que pacientes y familiares no serán capaces de acceder a ellas. Las páginas, procedidas por la palabra clave **users**, son:
 - **/add:** muestra un formulario para añadir nuevos usuarios al sistema. El gestor debe indicar un identificador (DNI, por ejemplo), un nombre completo, una contraseña y un rol. Para mantener la privacidad personal, se generará un identificador de usuario único a partir del valor real del identificador.
 - **/disable:** deshabilita un usuario, pero sin eliminarlo del sistema. Las relaciones del usuario se eliminan y, en el caso de los pacientes, se le desasignan todos los dispositivos que tenga asignados.
 - **/delete:** elimina completamente un usuario. Antes de poder eliminarlo, un usuario debe estar desabilitado.
 - **/enable:** activa un usuario deshabilitado anteriormente.
2. **Relaciones:** Procedidas por la palabra **relations**. Todas las páginas disponibles son:
 - **/:** mostrará las relaciones del usuario actual, en caso de que pudiera tenerlas (los administradores no pueden tener relaciones).
 - **/\$usuario:** relaciones del usuario especificado, si el usuario actual tiene los permisos suficientes.
 - **/\$usuario/add:** si se dispone de los permisos suficientes, muestra un formulario en el que se pueden escoger usuarios a los que añadir relación con el usuario especificado.
 - **/\$usuario1/delete/\$usuario2:** eliminará la relación entre el **usuario1** y el **usuario2**. De la misma manera que en el punto anterior, se deben disponer de permisos suficientes.

6.5 Informes de los pacientes

Los informes son datos que pueden ser creados para informar sobre el estado de un paciente en cierto momento. Estos informes pueden ser creados por el sistema multi-agente o también por otros usuarios relacionados (médicos, familiares o enfermeros). Para implementarlos, se ha desarrollado un plugin de nombre `rsReports`, que incluye todas las acciones, páginas, vistas y modelos necesarios para gestionarlos.

Los informes están basados en la clase `ElggObject`. Para diferenciar los informes de otras entidades, se ha fijado un nuevo subtipo para los objetos llamado `report`. En cuanto al contenido, este se almacena en el atributo `description` de la clase. Esto no es suficiente para representar todos los datos que se necesitan, que pueden llegar a ser muy complejos. Para solucionarlo, se incluyen tres metadatos: `reportSummary`, un pequeño resumen del informe; `reportAuthor`, entidad que ha generado el informe (persona o dispositivo); y `reportData`, datos adicionales para el informe codificados en una cadena de caracteres JSON, que pueden usarse para generar gráficos u otros esquemas. Esta última variable se detalla en la sección 8.1, donde se explican las funciones de la interfaz en la red social y, más específicamente, la función para añadir informes.

Código 6.4: Modelo para la gestión de informes en la red social

```

1 <?php
2 // Agregar un informe a un paciente con identificador guid
3 reports_add_report(
4     $guid,
5     $type,
6     $author_guid,
7     $summary,
8     $content,
9     $data
10 );
11 ?>

```

El modelo para la gestión de los informes es sencilla, ya que los informes **sólo se pueden añadir**, y no modificar o eliminar. Por tanto, sólo es necesaria una función con la cual añadir nuevos informes a un paciente. En el código 6.4 se puede observar cómo llamar a esta función. Si se observa el esquema de los *plugins* en la figura 6.2, se aprecia que la interfaz de la red requiere del módulo de informes. Esto se debe a que el MAS es capaz de crearlos de forma pro-activa. Para acabar, cabe recalcar que el modelo está implementado como una biblioteca de Elgg, por lo que antes de

poder usarlo hay que cargarla.

6.5.1 Páginas

El módulo de informes generará distintas páginas. Todas ellas estarán precedidas por la palabra clave `reports`. Estas páginas son sólo accesibles para usuarios que pueden tener relaciones. Es decir, no son accesibles para los secretarios o administradores. Las páginas son las siguientes:

- `/`: muestra la lista de informes que guardan alguna relación con el usuario. Es decir, los pacientes verán sus propios informes, mientras que los demás usuarios visualizarán los informes de sus pacientes relacionados. En caso de no existir informes o pacientes relacionados, se mostrará un error.
- `/$usuario`: presentará la lista de informes de un cierto usuario, siempre que se dispongan de los permisos necesarios.
- `/$usuario/see/$id`: expondrá un único informe de un usuario. Al igual que en el punto anterior, se deben contar con los permisos adecuados para visualizarlo.
- `/$usuario/add`: página para añadir un informe a un paciente de forma manual. Mostrará un formulario donde un usuario relacionado con el paciente podrá especificar el tipo de informe, un pequeño resumen y el contenido del mismo.

6.6 Dispositivos asistenciales

El generador principal de contenido en la red social son los dispositivos asistenciales asignados a los pacientes. La implementación de la red debe ofrecer algún modo de gestionarlos, es decir, crearlos, asignarlos o desasignarlos a los paciente y eliminarlos.

Un dispositivo se implementa mediante un `ElggObject` de subtipo `device`. Debido a que un dispositivo puede estar asignado a un número indefinido de pacientes durante su vida útil, el atributo `owner_guid` de la entidad se fijará al identificador del sitio. Esto implica que un dispositivo no tendrá ningún dueño. Además, cada dispositivo dispondrá un identificador de dispositivo (no el identificador dentro de Elgg) que se almacenará en el metadato de nombre `deviceID` y un metadato `data` que almacenará algunos datos que el dispositivo ha generado durante su uso.

Un mismo dispositivo **puede cambiar de paciente**. Para implementar esta relación se ha creado un nuevo tipo de relación entre ambos. Se ha creado un modelo para gestionar esta relación, la funciones del cual pueden verse en el código 6.5. Más en detalle, la relación consta de dos partes, dependiendo de la entidad principal. Por un lado, los pacientes disponen de una relación `has_device_assigned` que apunta a los dispositivos que tengan asignados. Por otro lado, un dispositivo asignado a algún paciente contará con una relación `assigned_to`, que señalará al paciente al cual está asignado.

Código 6.5: Modelo para la gestión de dispositivos en la red social

```

1 <?php
2 // Obtiene el paciente asignado a un dispositivo
3 devices_get_assigned_patient($device_guid);
4 // Obtiene todos los dispositivos asignados a un paciente
5 devices_get_assigned_to($patient_guid);
6 // Comprueba si un dispositivo esta asignado ya a un paciente
7 devices_is_assigned($device_guid);
8 // Asigna un dispositivo a un paciente
9 devices_assign_to($device_guid, $user_guid)
10 // Desasigna un dispositivo a un paciente
11 devices_unassign_from($device_guid, $user_guid);
12 // Obtiene los dispositivos asignados a algun paciente
13 devices_get_assigned();
14 // Obtiene los dispositivos libres (sin asignacion)
15 devices_get_unassigned();
16 ?>

```

6.6.1 Páginas

Para que los usuarios puedan visualizar y gestionar los dispositivos, el plugin ofrece diferentes páginas. Cada una de ellas vendrá precedida por la palabra `devices`. Las posibilidades son:

- `/`: mostrará los dispositivos del usuario, en caso de que este sea paciente.
- `/user/$usuario`: página con una lista de los dispositivos de un paciente. Dependiendo del rol y la relación del usuario actual y el paciente, este podrá asignarle dispositivos libres o también asignarle nuevos.

- `/user/$usuario/assign`: si se cuenta con los permisos necesarios, mostrará un formulario para asignar un dispositivo libre al usuario.
- `/user/$usuario/unassign/$device_guid`: desasignará el dispositivo indicado al usuario. De nuevo, se necesitan permisos para poder realizar esta acción.
- `/assigned`: lista de dispositivos que están asignados a algún paciente y el paciente al que están asignados. Sólo los gestores pueden ver esta lista.
- `/unassigned`: mostrará una lista de los dispositivos libres en el sistema. De igual manera que en el punto anterior, sólo los gestores pueden ver la lista.
- `/new`: formulario para añadir un nuevo dispositivo. El usuario podrá introducir un identificador para el dispositivo, el tipo de entre las opciones disponibles y la dirección del dispositivo. De nuevo, se necesitan los permisos suficientes.
- `/see/$device_guid`: visualización de un dispositivo concreto. Un usuario puede ver un dispositivo si es gestor de la red o si está asignado a él o a algún usuario relacionado.
- `/delete/$device_guid`: elimina un dispositivo que no esté asignado a ningún paciente. La página no genera ningún tipo de contenido. En su lugar, redirecciona a la lista de dispositivos al acabar de eliminar el dispositivo especificado. De nuevo, sólo los gestores de la red pueden eliminarlos.

6.7 Otros *plugins*

Los *plugins* de las secciones anteriores son la mayor parte del núcleo de la red social. Sin embargo, existen otros aspectos que hay que incluir en la red para cumplir los objetivos fijados en un principio y, además, completar la experiencia de los usuarios. En las siguientes secciones se describen las cinco modificaciones restantes.

6.7.1 rsActivity

Por defecto, Elgg muestra en la página principal la actividad de todos los usuarios registrados y permite, además, escoger mediante pestañas, ver la actividad de los amigos y la actividad personal. En este proyecto, como ya se ha visto, no existe la relación de amigos entre los usuarios, y no es una opción posible que todos los usuarios vean la actividad de los demás.

El objetivo de la modificación, por tanto, es modificar la página de actividad de los usuarios para que sólo se muestre la actividad de los pacientes relacionados. De esta manera, los doctores verán la actividad de los pacientes a los que están atendiendo, los familiares verán la actividad de sus parientes que están usando algún dispositivo, etc. Como anotación, los administradores de la red (secretarios incluidos) no pueden ver actividad, ya que no pueden tener relaciones. Sin embargo, estos usuarios verán una página básica en la que gestionar algunos aspectos de la página. Este punto se describe mejor en la sección 6.7.4, que especifica el *plugin rsManage*.

El *plugin* es muy sencillo, pero podría hacerse más complejo en un futuro. Por ejemplo, podría incluirse diferentes tipos de contenido dependiendo del rol de la persona que estuviera viendo la actividad.

6.7.2 rsProfile

El perfil de usuario que viene por defecto en Elgg es poco útil en la red social del proyecto. Este *plugin* reescribe esta página, que ahora muestra el identificador del paciente, su nombre completo e información del rol que desempeña.

Además, existen diferentes niveles de privacidad a la hora de mostrar información en el perfil. Por ejemplo, los usuarios relacionados podrán encontrar un enlace al listado de dispositivos e informes del paciente, mientras que su doctor, por ejemplo, podrá ver también un enlace para asignarle un nuevo dispositivo.

Al igual que otras de las modificaciones, esta queda libre para ampliar en un futuro, ya que se podría mostrar cualquier tipo de información interesante referente a la persona. En este proyecto se ha decidido, sin embargo, dejarlo lo más sencillo posible.

6.7.3 rsMessages

De nuevo, la relación de amigos por defecto en Elgg supone algunos problemas. En el caso de los mensajes, el módulo por defecto en el núcleo sólo permite enviar mensajes a amigos. En el caso de este proyecto, no obstante, se busca que los usuarios puedan enviar mensajes a cualquiera de los usuarios con los que guarda alguna relación.

La solución pasa por reemplazar la página que contiene el formulario para enviar mensajes. Esta página contiene un selector, donde se cargan al principio todos los amigos del usuario. Sólo hay que cambiar esta selección de amigos por la selección de usuarios relacionados, función que puede encontrarse en el modelo que controla los usuarios. Entonces, ahora se mostrarán sólo los usuarios relacionados. Además,

en la lista ahora se muestra el rol de esa persona, para que así la persona que envía el mensaje pueda saber qué relación existe entre ellos.

6.7.4 rsManage

Los usuarios que administran el sitio no pueden tener relaciones, como sí tienen los demás, pero necesitan una cuenta en la red social para gestionarla. Para solventar esto, esta modificación se encarga de ofrecer una página sencilla con algunos enlaces útiles de administración, que se mostrará como página principal para estos usuarios. Esta página mostrará enlaces para añadir nuevos usuarios y gestionar dispositivos. Además, también se redirigirá a esta si el usuario intenta acceder directamente a la página de actividad o a las páginas de informes de pacientes.

6.7.5 rsTheme

En este proyecto no se busca que la red social tenga un aspecto demasiado complejo. Sin embargo, hay mucha información innecesaria en el tema por defecto de Elgg, y otra información que debe añadirse. Por esa razón se decidió hacer un tema sencillo, que eliminara aquellos aspectos no útiles para el proyecto y que añadiera aquellos enlaces útiles a los menús dependiendo del usuario autenticado. Este tema está basado en un tema libre de la comunidad Elgg llamado `basic_light`.

Implementación del sistema multi-agente

El primer paso antes de empezar a implementar el sistema multi-agente es escoger la plataforma para los agentes y un sistema para implementarlos. Como ya se estudió en el capítulo 2, existe un estándar desarrollado por FIPA, y también diferentes implementaciones de este estándar en multitud de lenguajes de programación. Como no se busca ninguna característica en particular, se ha escogido la plataforma SPADE debido a estar implementada en Python, que hace el desarrollo más cómodo y rápido.

7.1 SPADE

SPADE es una plataforma para desarrollar sistemas multi-agente desarrollada en Python. SPADE cubre el estándar FIPA y soporta dos lenguajes de contenido: FIPA-SL y RDF. La plataforma utiliza un sistema de comunicación basado en mensajes XMPP [53]. Además, ofrece otras características predeterminadas como BDI o una interfaz Web para controlar los agentes. La plataforma acepta agentes desarrollados en otros lenguajes de programación, pero deben ser capaz de comunicarse con la plataforma mediante el protocolo XMPP. Por tanto, no es un problema si se busca integrar dispositivos con sistemas ya desarrollados. El único paso que habría que realizar en ese caso es adaptar el sistema para usar XMPP. Un esquema de la arquitectura usada por SPADE puede verse en la figura 7.1.

Una vez instalado SPADE y con la plataforma *online*, es posible crear agentes desde cualquier lugar. Para ello, sólo hay que crear un agente (del módulo `spade.Agent`) y ejecutar la función `start`. El agente más simple dispone de un nombre identificativo y una palabra secreta. Para realizar agentes más complejos se puede crear una nueva clase, que heredará del agente base de SPADE y podrá utilizar todas sus características.

7.1.1 Comportamientos

El componente principal de un agente en SPADE son los comportamientos. Los comportamientos [47] en SPADE son funciones que ejecuta el agente en determina-

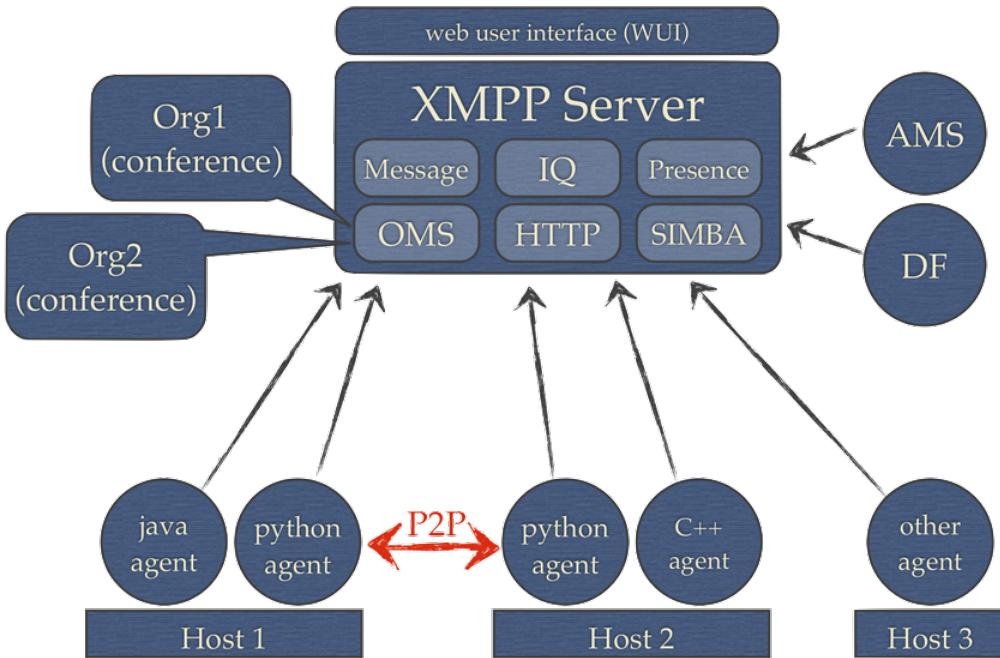


Figura 7.1: Esquema de la arquitectura de SPADE

das condiciones. Estos comportamientos pueden ejecutarse una sola vez, de forma permanente o después de determinados eventos. SPADE ofrece los siguientes tipos de comportamientos:

1. **OneShotBehaviour**: se ejecuta una sola vez y de forma instantánea cuando es añadido al agente.
2. **TimeOutBehaviour**: se ejecuta una vez, después de una cantidad fijada de tiempo después de añadirse al agente.
3. **PeriodicBehaviour**: se ejecuta indefinidas veces. Después de acabar una ejecución, espera una cantidad de tiempo predeterminada.
4. **EventBehaviour**: se ejecuta cuando sucede un cierto evento. Puede fijarse para que se ejecute una sola vez o cada vez que suceda el evento anterior.
5. **FSMBehaviour**: comportamiento complejo para trabajar con agentes con diferentes estados en un autómata finito determinista.

Un comportamiento puede añadirse a un agente especificando un **MessageTemplate**. Estas plantillas indican qué mensajes puede recibir el comportamiento cuando está ejecutándose, o cuándo se activa en caso de los comportamientos eventuales. Por tanto, un mismo agente puede tener varias instancias del mismo comportamiento.

ejecutándose con diferentes plantillas de mensaje, por lo que su comportamiento podrá ser diferente dependiendo de los mensajes que reciba el agente. Por tanto, un agente puede ejecutar un número indeterminado de comportamientos en un momento dado

7.1.2 Mensajes

Los agentes pueden enviar mensajes [48] a otros agentes. Estos mensajes serán recibidos por los comportamientos activos del agente que reciba el mensaje, y que cumplan con la plantilla establecida. Cada mensaje puede contener diferentes datos. Entre ellos, los más importantes son:

1. **Sender**: contiene información sobre el agente que ha enviado su mensaje. Es una instancia de la clase `AID`, que contiene el identificador del agente y una lista de sus direcciones.
2. **Performative**: los agentes pueden enviar un mensaje con diferentes objetivos [23]. Este atributo se basa en un *string* con una de las performativas del estándar FIPA-ACL [20]. Todas estas están definidas en la clase de SPADE `ACLMensaje`. Por ejemplo, si un agente quiere simplemente informar sobre algún hecho, que cree que el otro agente no conoce, utilizará la performativa `ACLMensaje.INFORM`. Por otro lado, si el agente necesita cierta información que desconoce, utilizará la performativa `ACLMensaje.REQUEST`.
3. **Content**: contiene una cadena de caracteres con el contenido del mensaje en sí, que el agente que recibe el mensaje tiene que decodificar y entender. En caso de que finalmente no lo entienda, puede responder al mensaje usando la performativa `ACLMensaje.NOT_UNDERSTOOD`.
4. **Encoding**: el contenido de un mensaje puede estar codificado de diferentes formas. En su versión más básica, el mensaje puede simplemente contener texto plano. Sin embargo, este contenido podría estar codificado de otra forma. Por ejemplo, podría contener datos codificados en JSON. Este atributo del mensaje puede servir al agente receptor para saber cómo decodificar el mensaje y entender su contenido.
5. **Ontology**: un mensaje puede pertenecer a diferentes tipos de conocimiento. Para que el agente receptor pueda saber a qué rango de conocimiento pertenece su consulta, este puede especificarlo en este parámetro. De nuevo, es tarea del agente receptor saber cómo actuar con la ontología.

7.1.3 AMS y DF

En el estándar desarrollado por FIPA existen dos agentes principales. Por un lado, el AMS (*Agent Management System*) se encarga de gestionar la presencia de los diferentes agentes que existen en la plataforma. Por otro lado, el DF (*Directory Facilitator*) se encarga de gestionar los diferentes servicios, que cada agente puede ofrecer en la plataforma.

El agente DF no es importante para este proyecto, ya que los agentes dispondrán del agente Manager (tal como se explicó en la sección 4.3.4). Este agente se comporta como una mezcla entre el AMS y el DF dentro del sistema. No obstante, el AMS sigue siendo obligatorio para que SPADE pueda controlar la presencia de los diferentes agentes. Aún así, SPADE gestiona este aspecto de forma automática. Cuando se crea un agente, este envía un mensaje de presencia al AMS. Después, cuando el agente finaliza, también se comunica automáticamente con él. Por tanto, la conexión con el AMS no es un problema siempre y cuando se utilicen agentes desarrollados con SPADE.

7.1.4 Base de conocimiento

SPADE ofrece dos formas con las que los agentes pueden guardar conocimiento [49]. La primera de ellas consiste en guardar hechos. Los hechos no son más que variables numéricas o cadenas de caracteres que un agente puede almacenar y consultar. La segunda consiste en guardar expresiones lógicas basadas en cláusulas de Horn. Es decir, el agente puede guardar expresiones del estilo $\neg p \vee \neg q \vee \dots \vee \neg t \vee u$, donde todos los literales son negativos excepto uno.

Esto permite a los agentes realizar operaciones con lógica de primer orden y deducir nuevo conocimiento. El código 7.1 muestra un pequeño ejemplo de cómo puede usarse este aspecto para obtener nuevo conocimiento utilizando implicaciones y conocimiento existente. En el código puede observarse que las expresiones que empiezan con minúscula representan variables libres, mientras que las que lo hacen en mayúscula representan constantes. Por tanto, $A(X)$ significa que la propiedad A se cumple para la constante X , mientras que $A(x)$ significa que la propiedad A se cumple para todo valor, que en ese momento se enlazaría con la variable libre x .

También hay que tener en cuenta que en el diseño del MAS se fijó una variable en el agente base para guardar datos. Esta se usará para guardar aquellos datos que quieran ser persistentes en el tiempo. Estos podrán ser enviados cada cierto tiempo a la red social, de tal forma que estarán presentes para el agente cada vez que se

reinicie el sistema.

Código 7.1: Ejemplo de deducción de conocimiento en SPADE

```

1 agent.addBelieve('A(X)')
2 agent.askBelieve('A(E)') # False
3 agent.askBelieve('A(X)') # True
4
5 agent.addBelieve('C(D)')
6
7 agent.addBelieve('F(x)')
8 agent.askBelieve('F(R)') # True
9 agent.askBelieve('F(X)') # True
10
11 agent.addBelieve('A(x) | C(x) -> B(x)')
12 agent.askBelieve('B(E)') # False
13 agent.askBelieve('B(X)') # True
14 agent.askBelieve('B(D)') # True

```

7.2 Jerarquía de ficheros

El sistema multi-agente estará integrado en un módulo de Python llamado `rssapm`. Dentro de este módulo se habrá tres submódulos con distintas funciones. Estos módulos son:

1. `agents`: contendrá la implementación de los agentes del núcleo del sistema.

Es decir, todos los agentes vistos en el diseño excepto la implementación específica de los dispositivos. Dispondrá de toda la jerarquía vista, desde los agentes más básicos hasta los más complejos. El primer agente heredará de `spade.Agent.Agent` (agente básico en SPADE), y los demás heredarán de este.

Este módulo está pensado para no tener que ser modificado por los desarrolladores si sólo se van a añadir nuevos dispositivos. Por esa razón, contendrá dos módulos abstractos para ayudar a los desarrolladores a generar nuevos dispositivos y sus comportamientos.

2. `devices`: contendrá la implementación de los servicios asistenciales (agentes y comportamientos). Este módulo está pensado para ser usado sólo por los desarrolladores, que implementarán clases heredadas de las plantillas vistas en

el módulo anterior (`agents`). Es decir, esta jerarquía pretende aislar el núcleo del sistema de la implementación de los dispositivos.

Deberá existir un módulo por cada dispositivo implementado. Dentro de ese módulo, los desarrolladores deberán implementar un agente para el dispositivo y comportamientos para el resto de agentes. Estos comportamientos se añadirán a los agentes que se especifiquen cuando el agente del dispositivo se inicie. Esto permitirá que la información generada por el dispositivo pueda ser procesada por todos los agentes necesarios. Para más detalle, estos aspectos se detallan mejor en el capítulo 9.

Hay que tener en cuenta que el agente del dispositivo se ejecuta en su propia máquina, mientras que los demás agentes y, por tanto los nuevos comportamientos, se ejecutan en el servidor. Esto implica que que la jerarquía deberá estar duplicada en ambas máquinas. Por un lado el dispositivo deberá disponer de la implementación del agente y, por otro lado, el servidor la de los comportamientos. No obstante, para facilitar el mantenimiento se recomienda disponer de todo el código en ambos lados.

3. `helper`: contiene clases para realizar diferentes funciones no relacionadas con los agentes. En este proyecto se ha añadido un módulo, de nombre `m1`, para hacer algunos cálculos de aprendizaje automático.

7.3 Agentes

En el directorio `agents` están implementados una serie de agentes con diferentes finalidades. En las siguientes subsecciones se resume su implementación y las funciones de cada módulo.

7.3.1 Agent.py

Agente básico del sistema del que heredará el resto de agentes. Hereda del agente base de SPADE `spade.Agent.Agent`. Una de sus finalidades es ofrecer herramientas para poder corregir errores en la implementación de los agentes que hereden de esta clase. Para ello, el desarrollador puede llamar a la función `set_h1_debug`, que fijará si el agente muestra avisos y errores de SPADE por la salida estándar. Además de esto, podrá producir sus propios mensajes de aviso o error, que se mostrarán de forma diferente a los proporcionados ya por SPADE. De esta manera se acelerará y dinamizará el desarrollo de los agentes para el sistemas.

La clase también ofrece algunas herramientas para tratar con mensajes. En concreto, dispone de dos funciones para tratar con conversaciones entre agentes, que son:

1. `reply_to_msg`: recibe un mensaje y una serie de atributos de un mensaje SPADE (`ACLMensaje`). A partir de ellos, crea una respuesta y la envía de nuevo al agente que envió el mensaje original.
2. `receive_reply`: toma como parámetros el mensaje del cual el agente quiere recibir una respuesta, el comportamiento que la está esperando y un tiempo de espera máximo. A partir del momento en que es llamada la función, esta espera una respuesta del mensaje pasado como parámetro (las respuestas mantienen un atributo llamado `conversationID`). Cuando recibe la respuesta, la redirige al comportamiento del agente que estaba esperándola. En caso de acabarse el tiempo, devuelve un valor nulo. SPADE tiene problemas para recibir mensajes en los comportamientos por evento. En ese caso, es posible lanzar un nuevo `OneShotBehaviour` para solucionar el problema.

Además de esto, el agente dispone de un diccionario donde puede guardar datos de todo tipo. Esto complementa al conocimiento que pueden guardar los agentes usando las herramientas de SPADE. La diferencia con este tipo de conocimiento es que está pensado para ser guardado de forma persistente en la red cada cierto tiempo. Es decir, el conocimiento de SPADE se reiniciará si se para el sistema, pero este no lo hará en aquellos agentes que existan en la red social.

Cada agente deberá decidir qué conocimiento debe ser persistente y cuál no. Para ello, se debe tener en cuenta que el conocimiento de SPADE se representa en forma de expresiones lógicas, lo que permite aplicar lógica de primer orden. El otro tipo de conocimiento, sin embargo, se basa en un diccionario simple de clave-valor. Un ejemplo de este conocimiento para el dispositivo de este proyecto (i-Walker) es la fecha del último ejercicio realizado. El agente guarda en cada ejercicio este valor para no procesar varias veces un ejercicio si el agente se reinicia. Por otro lado, también podría ser usado para guardar ciertas expresiones del conocimiento de SPADE de forma persistente, de tal forma que puedan ser recuperadas más tarde.

7.3.2 BDI`Agent`.py

Contiene las mismas variables que el agente base, pero además proporciona un comportamiento para utilizar el modelo BDI. Este comportamiento está implementado con un `PeriodicBehaviour` que, por defecto, se ejecuta cada segundo (aunque esto puede ser cambiado usando el parámetro `period` al crear el agente). Proporciona, además, cuatro funciones para añadir o eliminar objetivos al agente, que son atajos

a las funciones del comportamiento BDI. De esta manera, el desarrollador tiene que interactuar con un nivel menos de complejidad.

Cabe decir que SPADE ya dispone de una implementación muy completa del modelo BDI. No obstante, esta implementación está basada en servicios, que disponen de prerequisitos y variables de salida. SPADE utiliza las características de los servicios para componer planes, con los cuales conseguir los objetivos del agente. Estos servicios son funciones Python simples, y no comportamientos del propio agente. El problema de esta implementación es que SPADE recibe los mensajes de otros agentes no en el propio agente, sino en los comportamientos de este. Por tanto, dentro de un servicio existen problemas para comunicarse con los demás agentes del sistema, aspecto usado de forma notable en este proyecto.

Para solucionar el problema, se desarrolló el comportamiento BDI que contiene este agente. Por un lado, este comportamiento no es tan complejo como el que incorpora SPADE. Por ejemplo, el que viene por defecto puede generar planes compuestos a partir de diferentes servicios, mientras que el que ha sido desarrollado no. Por otro lado, no dispone de tareas reactivas o “normas”, mientras que el implementado para este proyecto sí que dispone de ellas.

En resumen, ambas implementaciones contienen puntos a favor y en contra. Sin embargo, la que se ha implementado en este proyecto se adapta completamente a las necesidades buscadas, aunque no se descarta incrementar el número y complejidad de sus funcionalidades para permitir agentes más complejos.

Cada vez que el agente ejecute el comportamiento BDI, este realizará una serie de comprobaciones, ejecutará una serie de acciones inmediatas y planificará acciones para hacer en los siguientes ciclos. Los pasos, en orden, son los siguientes:

1. **Comprobar objetivos activos:** es probable que se hayan producido acciones durante ciclo y ciclo que hayan afectado a alguno de los objetivos activos. Alguno de estos objetivos puede haberse completado ya o puede haber pasado a ser inalcanzable. En estos casos hay que desactivar el objetivo y fijarlo como le corresponda.
2. **Deliberar sobre nuevos objetivos:** el agente dispondrá de una lista de objetivos. Entre ellos, puede que algunos estén marcados como activos y otros como inactivos. Si un objetivo no está activo, el agente puede tener la intención de activarlo. Para ello, el objetivo que sea alcanzable y no haber sido completado ya. Existe, no obstante, una excepción. Los objetivos pueden ser persistentes en el tiempo, es decir, un objetivo persistente busca completarse tantas veces como sea posible, aunque se crea que este no es alcanzable o ya se ha completado anteriormente.

3. **Realización de acciones reactivas:** existirá una lista donde el agente dispondrá de varias tareas a realizar. Estas tareas tendrán unas precondiciones, lo que delimitará qué tareas realizar en uno u otro momento.
4. **Seleccionar intenciones:** existen tareas que no son reactivas y tienen, además de las precondiciones, unas postcondiciones. Estas tareas pueden ser usadas como planes para la consecución de los objetivos del agente. En esta etapa, el agente inspeccionará todas estas posibles tareas y mirará si es posible realizarlas para conseguir alguno de sus objetivos. Entre las que encuentre, escogerá aquella que tenga un menor número de efectos secundarios. Es decir, aquel plan que afecte menos a otros aspectos que no tengan que ver con su objetivo. Además, también se estudiarán las intenciones actuales para ver si siguen siendo factibles o deben ser eliminadas.
5. **Ejecutar intenciones:** las intenciones del agente se almacenarán en una cola, que ejecutará una intención por ciclo. Cuando se lleve a cabo, se eliminará de la cola. Así hasta que la cola quede vacía.

Como ya se ha comentado, el punto clave de este comportamiento son las tareas. Estas tareas se usan tanto para realizar acciones reactivas como para lograr objetivos. Esto se consigue haciendo que cada tarea pueda establecer unos prerequisitos y postrequisitos (salida). Debido a que los agentes usan diferentes formas de representar su conocimiento, el comportamiento BDI también incluye diferentes maneras para comprobar el conocimiento del agente a la hora de fijar una tarea. Para el proyecto se han incluido unas cuantas, que podrían ampliarse en un futuro. Estas formas son:

1. **expression:** si el parámetro de la condición es una cadena de caracteres, se comprobará la existencia o no existencia de la expresión en la base de conocimiento por defecto de SPADE.
2. **[‘G’, <expression>]:** comprueba si el agente dispone de un objetivo con la expresión establecida.
3. **[‘AG’, <expression>]:** de forma parecida al anterior, comprueba si el agente dispone un objetivo y si este está activo.
4. **[‘DE’, <name>]:** consulta si el agente dispone de una variable en su variable **data** (no en la base de conocimiento de SPADE).
5. **[‘D’, <op>, <name>, <value>]:** hace una comprobación booleana del valor de la variable **name** (campo **data** en el agente). Todos los operadores posibles son ==, >, <, >= y <=.
6. **[‘F’, <function>]:** para comportamientos más complejos, el comportamiento permite ejecutar funciones de comprobación personalizadas. La función debe

devolver un valor booleano y tener como parámetro un agente. A partir de esto, el desarrollador puede hacer cualquier comprobación accediendo a los datos del agente directamente.

En resumen, los agentes heredarán de `BDIAgent` si quieren seguir un modelo BDI, en el cual podrán fijar objetivos que buscan alcanzar y diferentes tareas, que pueden ser reactivas o para formar planes. La creencias del agente (*believes* en BDI) se llevarán a cabo mediante la base de conocimiento de SPADE y, en menor medida, con la variable `data` heredada del agente base.

7.3.3 Person.py

Clase de la que heredarán todos los agentes personales de la red social. A su vez, hereda de `BDIAgent`. Ofrece herramientas para guardar los datos persistentes de los agentes de las personas en la red social y también una función para enviar mensajes al agente gestor. Además de los datos contenidos por los agentes en su jerarquía, guarda las variables `username`, con el identificador de la red social; `guid`, el número identificador global en la red; y `role`, el nombre del rol que desempeña la persona.

7.3.4 Device.py

Plantilla para que los desarrolladores puedan generar agentes para dispositivos asistenciales. Estos agentes tienen un comportamiento BDI y disponen de algunos datos adicionales útiles para los dispositivos. Estos datos incluyen un nombre, compuesto por el identificador del paciente, el tipo de dispositivo y su identificador; una cadena de caracteres indicando el tipo de dispositivo; el identificador numérico global en la red social; y el nombre y dirección del agente del paciente para poder comunicarse con él.

Además de los datos adicionales, la clase ofrece tres funciones útiles para operar con los datos del dispositivo. La primera de ellas es `commit_data`, que guarda en la red social la información persiste del dispositivo. La segunda se trata de `sendMsgToPatient`, que facilita la comunicación del agente con su paciente. Por último, y al igual que el agente `Person`, también incluye la función `send_to_manager` para enviar mensajes al agente gestor.

Por último, la plantilla dispone de las herramientas necesarias para **comunicarse con el gestor de agentes automáticamente**. Cuando se inicie la actividad del dispositivo, este hará que inicie también su agente. Este agente, no obstante, no

conoce a qué paciente está asignado por lo que no puede enviarle los datos necesarios. Estas herramientas permiten que estos agentes se configuren de forma automática al inicio mediante el gestor de agentes, y a su vez que este pueda pararlo por cualquier razón. Por ejemplo, si el dispositivo es desasignado de su paciente, el agente de este será informado y parará su actividad. El dispositivo deberá iniciarse de nuevo cuando sea asignado a otro paciente, lo que hará que su agente se inicie y se vuelva a configurar correctamente. Por otro lado, es posible que el agente esté realizando procesos importantes cuando el gestor decida pararlo. Para poder darle un margen de maniobra en este caso, se incluye una nueva variable con nombre `can_stop`, que el desarrollador puede modificar para que no se pare el dispositivo directamente. En este caso, esperará hasta que la variable cambie de valor y, finalmente, se parará el dispositivo.

La configuración inicial del agente del dispositivo es invisible a los desarrolladores de nuevos dispositivos. Sin embargo, a la hora de añadir una nueva instancia del agente del dispositivo (sección 9.1) sí que se debe de tener en cuenta el identificador y la clave del dispositivo. Estos valores deberán ser los mismos que los introducidos en la red social, y el gestor los comprobará antes de mandar los datos necesarios para acabar la configuración. El paso seguido para autenticar el agente del dispositivo en el sistema puede verse en más detalle en la sección 2.

7.3.5 DeviceBehaviours.py

Los desarrolladores deben ser capaces de añadir comportamientos a los agentes relacionados con su dispositivo. Para ello, esta clase brinda el acceso a todos estos agentes, que se separarán entre el paciente y sus personas relacionadas. Al iniciarse un dispositivo, se recorrerán todas las personas relacionadas con este, y se iniciarán todos los comportamientos indicados por el desarrollador. De igual manera, este se encargará de pararlos cuando un dispositivo se elimine o se pare el sistema. Por tanto, debe llevar un control estricto de qué comportamientos se están ejecutando para saber si hay que proceder a pararlos o no.

Para lograr estos fines, el desarrollador dispondrá de cuatro funciones principales:

- `add_behavs_to_patient`: se ejecutará cuando se vaya a añadir comportamiento al agente del paciente. Este agente puede ser obtenido usando la variable `self._patient_agent`.
- `delete_behavs_from_patient`: igual que el anterior, pero de forma inversa. Se ejecutará cuando se elimine un dispositivo y haya que eliminar del paciente los comportamientos añadidos. Algunos de ellos pueden haber finalizado en este

momento, cosa que el desarrollador tiene que tener en cuenta.

- `add_behavs_to_related`: mismo paso que con el paciente, pero con un agente de un usuario relacionado. En este caso, el desarrollador recibirá el agente como parámetro, ya que pueden existir multitud de personas relacionadas con un solo paciente. Los comportamientos se añaden para cada agente manualmente. De esta manera, es posible añadir un comportamiento u otro dependiendo de alguna característica específica, como por ejemplo su rol.
- `delete_behavs_from_related`: misma acción que la anterior pero para eliminar los comportamientos. En este caso hay que tener más cuidado, ya que los comportamientos pueden ser muy diferentes dependiendo de cada agente.

7.3.6 Manager.py

Único agente obligatorio en el sistema, y que debe poseer un nombre conocido por todos los demás (`manager`). El agente gestiona la creación, manutención y eliminación de los demás agentes, por lo que es necesario para que todo funcione correctamente. La implementación no cambia en cuanto al diseño (sección 4.3.4). Los diferentes datos para guardar los agentes se han implementado de la siguiente manera:

- `_people`: diccionario¹ que guarda los agentes de las personas. La clave es el identificador numérico global en la red social. Cada entrada contiene un subdiccionario con tres elementos: `entity`, que almacena el agente de la persona y algunos datos útiles; y `groups` que representa una lista de todos los grupos en los que la persona está presente. De esta manera, si se elimina una persona del sistema, se puede eliminar su índice de los diferentes grupos de una forma eficiente.
- `_devices`: mismo tipo de diccionario, en este caso para guardar los diferentes dispositivos. Además de los elementos `entity` y `groups`, cada dispositivo contiene la variable `behaviours`, que almacena la instancia de la subclase de `DeviceBehaviours` creada por el dispositivo. De esta manera, si se añade una nueva persona a un grupo ya activo, el gestor de agentes le añadirá los comportamientos necesarios para cada dispositivo del grupo. Además de esto, los diccionarios poseen una pequeña diferencia respecto al diccionario de personas. Esta diferencia es que el campo `entity` no contiene el campo `agent` (la referencia a la instancia del agente), sino un campo `aid` que guarda el nombre y dirección del agente del dispositivo y otro `active` que indica si el agente

¹Estructura de datos que asocia una clave concreta con un valor determinado

ha iniciado. Esto sucede porque, a diferencia de las personas, los agentes de los dispositivos no están corriendo en la misma máquina que los agentes de personas. Por tanto, el agente **Manager** no tiene poder sobre sus instancias, ni puede iniciarlos o pararlos. Sólo puede esperar a que envíen un mensaje indicando que están iniciándose, identificarlos, guardar sus datos y enviarles la información necesaria.

- **_groups:** otro diccionario, en este caso para guardar los diferentes grupos. Su funcionamiento es algo diferente. La clave del diccionario es el identificador del paciente del grupo, y dentro de cada entrada hay un subdiccionario, que contiene tres claves: **patient** guarda la referencia al agente del paciente (identificador), **related** almacena un diccionario con las referencias de los agentes de personas relacionadas y **devices** hace lo mismo para los dispositivos del grupo.

La implementación de las funciones explicadas en el diseño del gestor no se van a detallar debido a que tienen una **longitud elevada**. Sin embargo, no cambian en lo referente al diseño original. Sólo tienen que tenerse en cuenta las estructuras de datos que se han visto y trabajar con ellas para que no se produzcan incongruencias. Esto puede suceder fácilmente a la hora de añadir o eliminar agentes si no se modifican los datos necesarios.

El otro aspecto que debe cubrir el gestor es el de proporcionar la información necesaria a los demás agentes cuando estos se lo pidan. Para ello, **Manager** dispone de un gestor de mensajes, implementado en forma de **EventBehaviour** y que se ejecuta cada vez que el agente recibe un mensaje. Cuando esto sucede, primero comprueba que el agente esté registrado en sus listas. Si lo está, entonces pasa a procesar su consulta. La consulta estará separada en diferentes segmentos, separados por dos puntos (:). Todas las posibles consultas serán:

1. **is_related:<nombre>**: si el agente que envía el mensaje y el otro agente están relacionados, enviará una respuesta con la performativa **ACLMensaje.CONFIRM**. En caso contrario, **ACLMensaje.DISCONFIRM**.
2. **get_related:<rol>**: obtendrá todas las personas relacionadas con un agente. El agente podrá indicar también (opcionalmente) el rol de las personas que está buscando mediante un segmento adicional. Se enviará una respuesta con la lista de los AID de los agentes, con la codificación **json_data**, es decir, un listado en forma de cadena de caracteres codificado con JSON.
3. **device.connecting:ID:key**: los dispositivos que estén iniciando sus agentes deben enviar este tipo de mensaje al **Manager**. Junto al mensaje deben especificar su identificador de dispositivo y su clave (que se encuentra cifrada mediante **SHA1**). El gestor buscará en el diccionario de dispositivos si estos

datos son correctos. En caso afirmativo, marcará la variable `active` de la entidad del dispositivo como cierta y guardará el AID del agente para poder consultarla después. Además, iniciará los comportamientos específicos para ese dispositivo en todos los agentes del grupo del paciente al que está asignado. Finalmente, responderá al mensaje del paciente con cierta información de utilidad para él. Esta información estará codificada en JSON y contendrá un diccionario con tres campos: `guid` (identificador numérico único del dispositivo), `patient_agent_name` (nombre del agente del paciente, para que pueda enviarle mensajes de forma directa) y `start_data` (datos persistentes del dispositivo que se guardan en la red social).

En caso de que la consulta no sea aceptada por alguna razón, el gestor contestará con una performativa `ACLMessages.REJECT_PROPOSAL`, y en el caso de que no pueda ser entendida, con una `ACLMessages.NOT_UNDERSTOOD`.

7.3.7 PatientAgent.py

El paciente es el agente más importante en cada grupo, ya que es él el que gestiona los datos de los diferentes dispositivos. Sin embargo, toda esta gestión se realizará mediante los diferentes comportamientos que añadirán los desarrolladores al añadir un nuevo dispositivo.

Por tanto, el agente del paciente debe almacenar información útil para estos comportamientos, y cierta información que pueda indicar su estado de ánimo dependiendo de las acciones que haya realizado mediante los diferentes dispositivos, si está seguro o si la red social necesita ser informada de algún aspecto.

El **conocimiento más importante** del paciente es la variable persistente `mood`, que establecerá en un rango de -10 a 10 el nivel de ánimo del paciente. Esta estimación podrá aumentar o disminuir dependiendo de los datos recogidos en los diferentes dispositivos y procesados por el agente. Por ejemplo, existen estudios [45] que muestran que caminar mejora el estado de ánimo. Esto es posible de medir con el caminador i-Walker, del cual se desarrollarán servicios de ejemplo. De esta manera, dependiendo de la distancia recorrida en un ejercicio que el paciente ha realizado en un momento y las recorridas en ocasiones anteriores, sería posible determinar si su estado de ánimo puede estar subiendo o bajando. De esta forma, existirá una tarea reactiva que avisará a los agentes relacionados con el paciente si se detecta que su estado de humor puede no ser bueno, o si por el contrario es muy bueno.

El estado de ánimo del paciente puede estar bajo en ciertas ocasiones. Se considera este límite cuando la estimación está por debajo de -7. En este caso, el sistema

multi-agente informará al resto de agentes relacionados sobre este aspecto. Estos agentes podrán decidir si alertar a sus usuarios en la red social o no. Se producirá un máximo de uno de estos avisos cada 24 horas hasta que el estado de ánimo mejore.

El resto de conocimiento se realizará mediante la base de conocimiento que ofrece SPADE. Este conocimiento incluirá posibles peligros, o si el agente necesita que se informe a su red de agentes sobre algún aspecto en concreto de su estado. Sería un buen punto a tener en cuenta para futuros proyectos estudiar qué otros aspectos puede almacenar el agente (que usen todos los dispositivos) y qué delegar a los comportamientos introducidos por cada dispositivo.

Además, el agente ofrece funciones para consultar los dispositivos asignados al paciente, para así poder comunicarse con ellos sin necesidad de consultar siempre con el **Manager**. Por tanto, guarda un diccionario de los dispositivos asignados, donde la clave es el identificador numérico global del dispositivo en la red social. A través de esta lista puede obtener los diferentes datos del dispositivo (agente, dirección, tipo) a través de su identificador o del nombre de su agente.

En resumen, se ha mantenido un esquema de agente simple. Este sólo guarda cierta información de su estado, que puede ser consultada y modificada por un número indefinido de dispositivos asignados al paciente. Eso significa que la gran mayoría del procesado de información se realizará por los comportamientos desarrollados para los dispositivos, que se acoplarán al agente del paciente. Estos comportamientos son los únicos que conocerán cómo se comportan los datos recibidos por cada tipo de dispositivo. No conocerán, por tanto, los datos específicos de los demás, sino que conocerán cómo han afectado teóricamente al estado de ánimo del paciente. Mediante este dato podrán interpretar y procesar los datos de su dispositivo de una u otra forma.

7.3.8 Otros agentes personales

El resto de agentes personales se han implementado de una manera simple y reactiva. En el ámbito de este proyecto, estos agentes sólo reaccionan ante mensajes del paciente y actúan sobre las personas a los que representan en la red social dependiendo de estos mensajes. En este proyecto, estos agentes sólo envían notificaciones a la red cuando reciben un aviso del agente del paciente.

Para implementarlos, estos heredan del agente **Person**. Todos disponen de un gestor de mensajes, que se activa cuando reciben cualquier mensaje. Una vez recibido el mensaje, preguntan al gestor sobre la procedencia de este. En caso de que provenga de uno de los pacientes con los que está relacionado, procesarán el mensaje en

busca de alertas. Estas alertas pueden haber sido generadas por un dispositivo o por el propio paciente. En cualquier de los dos casos, el agente responderá con una performativa `ACLMensaje.AGREE`, o `ACLMensaje.NOT_UNDERSTOOD` en caso de no entender el contenido del mensaje.

Aunque se ha decidido que estos agentes serán simples, la puerta queda abierta a los futuros desarrollos para que incrementen la cantidad de funcionalidades de estos agentes, incluyendo todo lo que sea necesario al código, que no está limitado de ninguna manera. Aún así, un desarrollador podrá añadir todos los comportamientos que sean necesarios a estos agentes para que estos gestionen los datos de un dispositivo específico. No obstante, los servicios de ejemplo de este proyecto (caminador i-Walker) solo disponen de comportamientos del lado del paciente, ya que los comportamientos básicos de los demás agentes ya proporcionan las funcionalidades necesarias para cumplir con los objetivos. Otros comportamientos que podrían ser añadidos a los dispositivos podrían ser:

- Procesado más complejo de los datos. En este proyecto, los agentes de doctores, familiares, etc. sólo se encargan de mantener informados a sus usuarios en la red social a través de los datos procesados por el agente del paciente y retransmitidos a los demás. Podría ser interesante para algunos dispositivos que este procesado fuera diferente para cada uno de los agentes, produciendo diferentes resultados.
- Configuración por parte de los médicos. Los dispositivos podrían necesitar de cierta configuración. Esta podría realizarse de forma automática a partir del conocimiento del agente del médico que trata al paciente. Cada dispositivo es libre de añadir este tipo de comportamientos a los doctores. Para el dispositivo de este proyecto, no obstante, la configuración se debe realizar de forma manual antes de empezar un ejercicio. Este aspecto sería una buena idea para introducir en un futuro.

Implementación de la interfaz

Una vez desarrollados los dos sistemas principales, estos deben poder **comunicarse** de alguna forma. Esta conexión debe ofrecer las funcionalidades suficientes para lograr los diferentes objetivos del proyecto, y además debe ser **extensible** para poder aumentar sus funcionalidades en un futuro. De forma contraria, ambos sistemas podrían ser modificados pero no podría haber comunicación de estas nuevas modificaciones entre los dos. Además, la interfaz debe ser bidireccional, es decir, ambos sistemas deben poder consultar al otro.

La implementación de la interfaz presenta un problema inicial. ¿La interfaz debería ser un sistema aislado o estar incluído en los sistemas? El único requisito para que la interfaz sea plausible pasa porque funcione en un medio y lenguaje que entienda ambas partes. Por un lado se encuentra el sistema multi-agente, que usa la plataforma SPADE usando Python. Por otro lado está la red social, implementada en un servidor Web usando Apache y PHP.

Existen diferentes formas en las que ambos sistemas podrían comunicarse, que tienen sus ventajas y desventajas. La primera de ellas consistiría en usar los ficheros del propio sistema operativo, los cuales serían leídos por la interfaz, que realizaría las acciones necesarias. Este método tiene diferentes lagunas. En primer lugar, los tipos de fichero podrían variar dependiendo del sistema operativo, por lo que la programación podría complicarse. En segundo lugar, sería un problema en caso de escalar los sistemas, ya que podrían estar ejecutándose en diferentes máquinas. En ese caso, habría que usar, por ejemplo, un servidor FTP¹ para la comunicación, algo que elevaría la complejidad de forma innecesaria.

Otra forma posible de comunicación es usar algún tipo de protocolo de transferencia de datos en red. En este caso la elección es sencilla. Se puede utilizar HTTP, ya que ambos lenguajes de programación pueden enviar peticiones con este protocolo. Una vez fijado el protocolo de transferencia, toca fijar el lenguaje de comunicación. Esto se debe a que HTTP es la sigla para *Hypertext Transfer Protocol*, es decir, los sistemas podrán enviar texto, no objetos del propio lenguaje (enteros, *arrays*, diccionarios, etc.). Además, cada lenguaje puede tener sus particularidades. Por tanto, los datos que se envíen a través de la interfaz deben ser transformados a texto por el lenguaje

¹Protocolo para la transmisión de datos en red

que envía los datos y después transformados de nuevo a objetos del lenguaje que los reciba. Las dos opciones en este caso fueron XML y JSON. Ambos pueden ser entendidos por los dos lenguajes, pero JSON es más amigable. En el caso de PHP, se pueden usar las funciones predeterminadas `json_encode` y `json_decode`. En el caso de Python, primero hay que importar el módulo `json` y, después, se pueden usar las funciones `json.dumps` y `json.loads`.

Por tanto, una parte de cada sistema será el encargado de enviar peticiones a la interfaz usando HTTP y JSON. El siguiente problema, sin embargo, es dónde enviar estas peticiones. ¿Existirá un sistema aislado que manejará las peticiones de ambos o puede realizarse de forma directa? En este aspecto se puede dar uso a una de las características de Elgg. El motor de redes sociales permite “exponer” funciones de los diferentes *plugins* de la red, permitiendo a terceros sistemas acceder a estas funciones. Por tanto, la red social ya tiene un servidor para gestionar las peticiones recibidas. Debido a esto, se decidió que la comunicación entre sistemas sería directa, por lo que se debía implementar también un servidor de peticiones en el MAS.

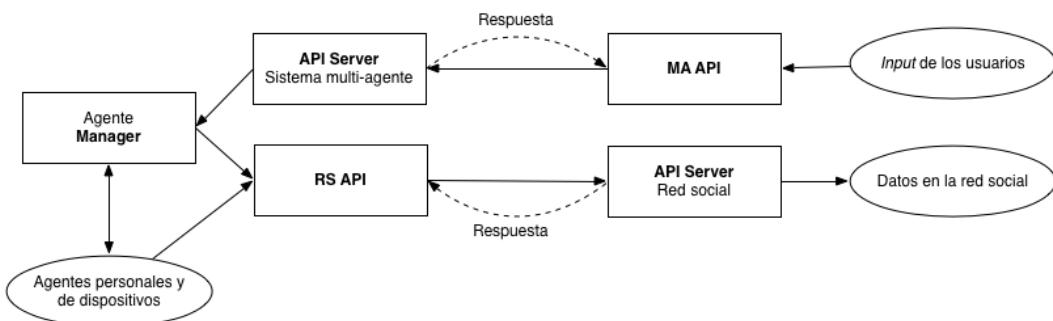


Figura 8.1: Esquema de la interfaz entre red social y sistema multi-agente

En el esquema de la figura 8.1 pueden apreciarse las diferentes partes de la interfaz. En el taba, cada sistema dispondrá de un servidor para manejar las peticiones y un módulo para enviarlas. Algunas de las posibles interacciones que pueden darse son:

- Un usuario realiza alguna acción en la red social que hace necesario informar al sistema multi-agente. El controlador que gestiona esa acción mandará la petición al módulo encargado de enviar las peticiones. Este último se encargará de enviarla al servidor del MAS y recibir la respuesta, que procesará y devolverá al controlador.
- El gestor de agentes (**Manager**) o otro agente delibera que es importante informar a la red social sobre algún aspecto. En este caso, el agente llamará a la función adecuada del módulo que comunica con la red social. Al igual que antes, este envía la petición al servidor de la red y redirigirá la respuesta procesada.

En las siguientes secciones se detalla mejor el funcionamiento de cada uno de los servidores receptores de solicitudes y todas las posibilidades que ofrecen. Además, se establecen algunas medidas de seguridad, para que la interfaz pueda ser usada sólo por aquellos sistemas autorizados.

8.1 Peticiones en la red social

Elgg dispone de un sistema para exponer funciones al exterior de la red. Estas funciones pueden exponerse en cualquiera de los *plugins* activos. No obstante, se decidió agrupar todas ellas en uno solo, que toma el nombre `rsAgentAPI`. Este módulo tiene dos bibliotecas: `maAPI.php`, que tiene el objetivo de enviar peticiones al servidor del sistema multi-agente, y `API.php`, que contiene todas las funciones a exponer en el servidor de la red.

Por cada acción expuesta se debe especificar su nombre, parámetros y qué tipo de acción va a realizar. Cada parámetro debe disponer de un nombre y un tipo. Elgg rechazará la ejecución de ese método si el número de parámetros y sus tipos no es correcto. El tipo de acción incluye todos los métodos de HTTP: GET, para obtener datos; POST para actualizar datos; DELETE, para eliminarlos y PUT para crearlos. Para ahorrar complejidad, sin embargo, se usará sólo GET para obtener datos y POST para trabajar con ellos.

El MAS podrá acceder a estas funciones mandando peticiones HTTP a Elgg con una dirección específica. En concreto, la construcción de la dirección es, partiendo de la raíz de Elgg: `services, api, <formato>, json, ?method=<método>`. El formato puede ser escogido por el otro sistema entre XML y JSON. Aún así, ya se especificó anteriormente que ambos harían uso de la notación de Javascript. La respuesta a la petición es un diccionario con dos atributos: `status` y `result`. El primero de ellos indica el estado en el que ha acabado la petición: -1 si se ha producido algún error y 0 si se ha ejecutado correctamente. El segundo es la salida de la propia función en caso de que no haya error o el error en sí.

Las funciones expuestas por la red deben permitir al sistema multi-agente obtener los datos necesarios de los usuarios y los dispositivos de los pacientes. Además, también debe ser capaz de **crear nuevo contenido de forma proactiva**. A continuación se describen todas las funciones disponibles y su utilidad. Podría ser una buena idea para proyectos futuros ampliar este abanico de funciones para permitir nuevas funcionalidades, ampliando la cantidad de acciones que puede realizar el MAS. La exposición de todas estas funciones se puede observar en el código 8.3, que muestra de una forma más precisa los parámetros necesarios para cada función. En el código

puede apreciarse como todas las funciones tienen un parámetro `key`. Se trata de un parámetro genérico en las funciones para poder identificar al sistema que envía la petición, para así evitar problemas de seguridad. Esta clave es única y conocida sólo por ambos sistemas.

1. `users`: devuelve un listado de usuarios de la red. No acepta ningún parámetro para el filtrado. La salida JSON es un `Array` de `Object` con los atributos `username`, `guid` y `role` de cada usuario.
2. `users.groups`: devuelve una lista de grupos paciente-dispositivos-relaciones, es decir, por cada paciente se muestran sus dispositivos y los usuarios relacionados con él. Al igual que la función anterior, no acepta parámetros. El formato de salida es algo más complicado. Un ejemplo puede encontrarse en el código 8.1.

Código 8.1: Formato de salida de la función para obtener grupos de la red

```

1  [
2    {
3      "patient": {
4        "username": "Paciente",
5        "guid": 2,
6        "role": "PATIENT_ROLE"
7      },
8      "devices": [{"guid": 4, "type": "walker", "deviceID": "41"}],
9      "related": [
10        {"username": "Doctor", "guid": 1, "role": "DOCTOR_ROLE"},
11        {"username": "Familiar", "guid": 3, "role": "RELATIVE_ROLE"}
12      ]
13    }
14  ]

```

3. `users.reports.new`: genera un nuevo informe para un paciente. Es una acción POST que acepta los siguientes parámetros: `guid` (identificador del usuario), `type` (tipo del informe: information, warning o alert), `author` (identificador del autor del informe, o NULL), `summary` (resumen del informe), `content` (contenido completo) y `data` (datos complementarios codificados en JSON).

El campo `data` permite generar gráficos dentro de los informes, aunque podría ser usado para generar otro tipo de contenido. Para que los datos incluídos pararezcan dentro del texto del informe, se deben incluir variables dentro de este. La red social se encargará después de substituirán estas variables por el contenido adecuado con los datos específicados. Por ejemplo, si se especifica la variable `#{{chart}}` dentro del contenido de un informe, el sistema

intentará buscar datos dentro del parámetro `data` con nombre `chart`. En el código 8.2 hay un ejemplo de este parámetros para generar un histograma sencillo.

Código 8.2: Ejemplo del parámetro `data` para la función `users.reports.new` (generación de gráficos)

```

1  [
2    {
3      "type": "chart",
4      "name": "chart",
5      "data": [
6        {
7          "values": [
8            {"x": "tiempo_en_milisegundos_1", "y": 5.2},
9            {"x": "tiempo_en_milisegundos_2", "y": 4.6},
10           {"x": "tiempo_en_milisegundos_3", "y": 3.2},
11         ],
12         "key": "Distancia"
13       }
14     ],
15     "metadata": {
16       "type": "column",
17       "xAxis": "Tiempo",
18       "xAxisTime": true
19       "yAxis": "Km"
20     }
21   }
22 ]

```

Este tipo de parámetro es muy dinámico y puede servir para desarrollar nuevas funcionalidades en el sistema. No obstante, en este proyecto sólo se incluyen gráficos sencillos. Para ello, un gráfico debe especificar los siguientes campos:

- `type: chart`
- `name:` nombre del gráfico. Se usará para buscar una variable con este nombre en el contenido del informe, que será sustituida por el propio gráfico.
- `data:` *array* con un listado de series para el gráfico. Cada serie debe disponer de los atributos `key` (nombre de la serie) y `values`, un array con atributos `x` e `y` para cada punto del gráfico.

- **metadata:** atributos que definen el gráfico. El único campo obligatorio es **type**, que dispone de las opciones **column**, **line** o **area**. Además, existen otros campos opcionales: **xAxis** (título del eje x, cadena de caracteres), **xAxisTime** (fija el eje x como temporal, valor booleano), **yAxis** (título del eje y, cadena de caracteres).
4. **users.notifications.new:** algunas situaciones en los dispositivos pueden no ser suficientes para generar un informe, pero sí para generar una alerta. Por ejemplo, si se detecta que un paciente ha podido sufrir una caída, el sistema multi-agente podría decidir avisar a su médico o a sus familiares. Esta función se encarga de ello. Para lograrlo, se piden cinco parámetros al sistema: **to_guid**, identificador del usuario al que se añadirá la notificación; **author_type**: tipo de entidad que ha generado la notificación, puede ser **device** para notificaciones generadas por dispositivos o **person** si para aquellas generadas directamente por el sistema multi-agente; **author_guid**, entidad que ha producido la alerta (un dispositivo, por ejemplo); **type**, tipo de notificación; **entity_guid**, identificador del contenido de la alerta o identificador del usuario para que se dirija a su perfil; **description**, el texto que aparecerá en la notificación.

Además de los parámetros anteriores, el campo **description** puede contener variables en su interior. Estas variables servirán para mostrar en la notificación algunos contenidos de interés que no dispone el sistema multi-agente. Las variables se representarán de la siguiente manera: `#{{variable}}`. Se han incluido unas pocas variables de utilidad para los fines del proyecto, aunque podrían incluirse nuevas. Estas variables son:

 - a) **to_user_id:** identificador en la red del usuario al que se manda la notificación.
 - b) **to_user_name:** nombre completo del usuario que recibe la notificación.
 - c) **device_id:** identificador del dispositivo que ha generado la notificación, siempre que **author_type** sea **device**.
 - d) **from_user_id:** identificador en la red del usuario que hace referencia el contenido, si **entity_guid** hace referencia a un usuario.
 - e) **from_user_name:** nombre completo del usuario especificado en el elemento anterior.
 5. **devices.update.data:** los dispositivos en la red, además de contar con sus datos básicos, disponen de un metadato de nombre **data** que contiene algunos datos de interés generados durante sus ejercicios. En un momento dado, el sistema multi-agente podría decidir que esta información debe quedar reflejada

de forma permanente en la red. Para ello, se debe especificar el guid del dispositivo y los datos en el parámetro `data`, codificados en JSON.

6. `users.update.data`: al igual que en el punto anterior, los usuarios también disponen de datos adicionales. El funcionamiento de esta función es igual que antes, pero indicando un identificador de un usuario en la red social.

Código 8.3: Funciones en la interfaz de la red social

```

1  <?php
2  expose_function(
3      'users', 'api_users',
4      array(
5          'key' => array('type' => 'string')
6      ),
7      'Obtiene una lista de todos los usuarios',
8      'GET', false, false
9  );
10 expose_function(
11     'users.groups', 'api_get_groups',
12     array(
13         'key' => array('type' => 'string')
14     ),
15     'Obtiene todos los grupos (paciente, relaciones) de la red',
16     'GET', false, false
17 );
18 expose_function(
19     'users.reports.new', 'api_new_report',
20     array(
21         'key' => array('type' => 'string'),
22         'guid' => array('type' => 'integer'),
23         'type' => array('type' => 'string'),
24         'author' => array('type' => 'integer'),
25         'summary' => array('type' => 'string'),
26         'content' => array('type' => 'string'),
27         'data' => array('type' => 'string'),    // codificado en JSON
28     ),
29     'Genera un nuevo informe de un usuario',
30     'POST', false, false
31 );
32

```

```
33 expose_function(
34   'users.notifications.new', 'api_new_notification',
35   array(
36     'key' => array('type' => 'string'),
37     'to_guid' => array('type' => 'integer'),
38     'author_type' => array('type' => 'string'), // device o person
39     'author_guid' => array('type' => 'integer'),
40     'type' => array('type' => 'string'),
41     'entity_guid' => array('type' => 'integer'),
42     'description' => array('type' => 'string')
43   ),
44   'Genera una nueva notificacion',
45   'POST', false, false
46 );
47
48 expose_function(
49   'devices.update.data', 'api_update_device_data',
50   array(
51     'key' => array('type' => 'string'),
52     'guid' => array('type' => 'integer'),
53     'data' => array('type' => 'string'), // codificado en JSON
54   ),
55   'Actualiza los datos de un dispositivo de la red',
56   'POST', false, false
57 );
58
59 expose_function(
60   'users.update.data', 'api_update_person_data',
61   array(
62     'key' => array('type' => 'string'),
63     'guid' => array('type' => 'integer'),
64     'data' => array('type' => 'string'), // codificado en JSON
65   ),
66   'Actualiza los datos de una persona de la red',
67   'POST', false, false
68 );
69 ?>
```

8.2 Peticiones en el sistema multi-agente

Lo primero que hay que pensar antes de poder implementar la interfaz en el sistema multi-agente usando Python es el servidor HTTP para atender las peticiones. Python ofrece la posibilidad de crear un servidor simple con `SimpleHTTPServer`, pero se busca uno con el que crear una RESTful [44] API parecida a la de Elgg de la forma más sencilla.

El módulo `Bottle` ofrece las características suficientes para realizar una API completa. De forma parecida a lo visto en la sección anterior, se establecerá una sola página en la que la red social hará las peticiones. Esta página necesitará, al menos, los parámetros `method` para fijar la función a ejecutar y `key` para autenticarse en el sistema. Cualquier acceso a una página distinta a la principal generará un error HTTP 404.

Una vez enviada una petición a la página principal, se realizarán diferentes comprobaciones. La primera consiste en verificar la clave de acceso. Si la clave no existe o es incorrecta, no se deja acceso a más información que esa. Una vez autenticado, se procede a buscar el método (parámetro `method` en la consulta). Si este no existe o no es válido, se produce un error indicando que el método no existe. El siguiente paso consiste en comprobar que el método de llamada (GET o POST) es el correcto. Para acabar, sólo queda comprobar que el resto de parámetros sean los adecuados en número y tipo. En ese momento ya es posible llamar a la función correspondiente, que podrá generar también sus propios errores.

El sistema multi-agente debe permitir que la red social modifique los agentes de personas y dispositivos, cree nuevos grupos o elimine los que ya existen. Por ejemplo, si un médico indica que ahora está relacionado con un paciente (este pasará a ser su médico), esto debe verse reflejado en los agentes sin que sea necesario reiniciar los sistemas. Para ello, la interfaz hace uso del agente `Manager`, que tiene el poder de gestionar (iniciar, parar o generar) a los demás agentes. A continuación se detallan todas las funciones existentes, que podrían ser ampliadas en el futuro. Además, el código 8.4 muestra el tipo de llamada de cada método y el número y tipo de sus parámetros.

- `devices.add`: asigna un nuevo dispositivo a un paciente existente. Recibe como parámetros su identificador en la red (`guid`), su identificador real (`deviceID`), tipo (`type`), clave (`key`) y los datos de su paciente (`patientGUID` y `patientName`). El dispositivo se añadirá al grupo del paciente y se marcará el dispositivo como inactivo hasta que empiece a enviar datos.
- `devices.delete`: elimina un dispositivo ya presente en el sistema. Si estaba

activo anteriormente, primero eliminará todos los comportamientos que definió. Recibe un solo parametro, el identificador del dispositivo en la red (guid).

- **person.add**: añade una nueva persona al sistema. Recibe los parámetros **guid** (identificador numérico), **patient_guid** (identificador del paciente relacionado). En caso de que sea un paciente será igual al parámetro anterior), **type** (rol de la persona) y **username** (nombre para el agente). Al añadirlo al grupo, el gestor crea su agente e inicia su ejecución.
- **person.add.group**: función parecida a la anterior, pero en este caso sólo se indican su identificador y el identificador del nuevo grupo. Esta función se usa cuando la persona ya está presente en el sistema.
- **person.delete.group**: elimina a una persona de un cierto grupo. Recibe los mismos parámetros que la función anterior.
- **person.delete**: elimina a una persona del sistema completamente. Es decir, elimina al agente de todos sus grupos y después para y elimina el agente en sí. En caso de que sea un paciente, elimina por completo su grupo. Para lograrlo, sólo necesita su identificador en la red (guid) como parámetro.

Código 8.4: Referencia de las funciones expuestas por el sistema multi-agente

```

1  self._methods = {
2      'devices.add': {
3          'call_methods': ['POST'],
4          'method': self.add_device,
5          'args': {
6              'guid': int,
7              'deviceID': str,
8              'deviceKey': str,
9              'type': str,
10             'patientGUID': int,
11             'patientName': str
12         }
13     },
14     'devices.delete': {
15         'call_methods': ['POST'],
16         'method': self.delete_device,
17         'args': { 'guid': int }
18     },
19     'person.add': {
20         'call_methods': ['POST'],

```

```

21     'method': self.add_person,
22     'args': {
23         'guid': int,
24         'patientGUID': int,
25         'type': str,
26         'username': str
27     }
28 },
29 'person.add.group': {
30     'call_methods': ['POST'],
31     'method': self.add_person_to_group,
32     'args': {
33         'guid': int,
34         'patientGUID': int
35     }
36 },
37 'person.delete.group': {
38     'call_methods': ['POST'],
39     'method': self.delete_person_from_group,
40     'args': {
41         'guid': int,
42         'patientGUID': int
43     }
44 },
45 'person.delete': {
46     'call_methods': ['POST'],
47     'method': self.delete_person,
48     'args': { 'guid': int }
49 }
50 }
```

8.3 Seguridad

Otro de los puntos a tener en cuenta es la seguridad. Algunas de las acciones de la interfaz vistas permiten alterar los sistemas de **forma drástica**. Por un lado, Elgg ofrece un sistema de *API keys* sofisificado, pero dado que no es necesario distinguir entre diferentes sistemas accediendo a la interfaz, se ha decidido incluir una única clave *hard-coded*. Esta clave será conocida sólo por los dos sistemas, que denegarán

la ejecución de las peticiones si la clave no es correcta o no se ha especificado. Si no existiera o fuera incorrecta, no se realizará dicha acción y se generaría una respuesta con el error.

Por otro lado, se podría incluir una directiva en el fichero `.htaccess` [3] del servidor Apache de la red social para que cualquier dirección externa no pueda acceder a los servicios Web. Esta medida de seguridad, claro, debería ser eliminada en un futuro si fuera necesario separar ambos sistemas en diferentes equipos. En ese caso, podría incluirse una directiva para permitir sólo la dirección del otro servidor. Por otro lado, también se podría llegar a ofrecer una API más restrictiva a otros desarrolladores. En ese caso, se debería usar un sistema de claves más complejo y eliminar la directiva de restricción de direcciones.

Otro aspecto importante sería cifrar los datos transmitidos. En concreto para España, la Ley Orgánica de Protección de Datos de Carácter Personal (sección 12.5.4) establece que los datos que contengan información médica deben ser transmitidos de manera que no sea posible la escucha o modificación por parte de terceros. No obstante, este aspecto ha quedado fuera del alcance del proyecto, aunque debería ser implementado a la hora de llevar los sistemas a producción.

Implementación de los servicios: i-Walker

La implementación de los servicios basados en el caminador i-Walker seguirá las pautas especificadas en el diseño. Es decir, el dispositivo dispondrá de dos secciones bien diferenciadas. La primera de ellas será el agente del dispositivo, que captará y hará un proceso de los datos de los diferentes sensores del caminador. Este agente se ejecuta en el *hardware* del dispositivo y, mediante la red, enviará estos datos al agente del paciente al que esté asignado. La segunda parte añade los comportamientos necesarios a los agentes relacionados con el paciente asignado al dispositivo. De esta forma se puede hacer otro procesado de los datos enviados y distribuirlos a la red social. En este caso, los comportamientos se ejecutan en el servidor (máquina donde se encuentra el gestor de agentes). Por esta razón, el dispositivo deberá disponer del código del agente y el servidor del código de los comportamientos. En las diferentes secciones de este capítulo se detallan diferentes aspectos de la implementación de ambos.

9.1 Jerarquía de directorios

Uno de los objetivos del proyecto es dejar la puerta abierta a futuros desarrolladores para **implementar nuevos servicios**. Para conseguir facilitar este trabajo, no obstante, la jerarquía de los diferentes directorios y ficheros debe seguir una estructura bien definida de antemano. De esta manera, los desarrolladores sólo deberán colocar su código en el sitio adecuado y generar un pequeño archivo con la configuración de los dispositivos que residen en esa máquina.

En la raíz del sistema multi-agente hay diferentes *scripts* para iniciar el sistema. Uno de ellos toma el nombre de `start_devices.py`. Este *script*, primero, carga todas las clases de agentes de dispositivos que han sido implementadas. Después, acude al fichero de configuración (`devices.json`), que contiene un listado de dispositivos con su información relevante (identificador, tipo y clave de acceso). Mediante este listado y las diferentes clases, crea un agente por cada dispositivo e inicia su configuración. En el código 9.1 puede apreciarse un ejemplo del fichero `devices.json`. En este ejemplo existe un solo dispositivo, de tipo `walker` e identificador `wlk-42`. Esto hará que, a la hora de iniciar el sistema, el *script* cree un único agente, que representa a un

caminador i-Walker, de nombre `wlk-42@<dirección_servidor>`.

Código 9.1: Ejemplo del fichero de configuración de dispositivos

```

1  [
2    {
3      "ID": "wlk-42",
4      "type": "walker",
5      "key": "NspGrCsnMO"
6    }
7 ]

```

Para lograr hacer funcionar el agente y los comportamientos, el desarrollador deberá crear al menos dos ficheros. El primero contendrá la clase del agente, que heredará de `rssapm.agents.Device`. El segundo contendrá una clase con los diferentes comportamientos, tal como se estipula en la clase `rssapm.agents.DeviceBehaviours`, de la que heredará esta. Estos ficheros deberán colocarse en un directorio, preferiblemente nombrado igual que el dispositivo, dentro del directorio `rssapm/devices`. En ese módulo podría existir una nueva jerarquía completa, todo depende de la complejidad del dispositivo. No obstante, en la raíz de esta deberá existir un fichero `__init__.py` que genere dos variables: `config_agents` y `config_behaviours`. Cada una de estas variables será un diccionario, donde la clave será el nombre del dispositivo y el valor la clase pertinente (agente o comportamientos). Aunque ambas partes deberán ejecutarse en diferentes máquinas y, por tanto, no se ejecutarán ambas clases a la vez, se recomienda implementar las dos en el mismo módulo y después copiar los ficheros a las diferentes partes. En el código 9.2 hay un ejemplo de este fichero para el i-Walker. Primero se importan los módulos que contienen ambas clases, para finalmente asignarlas al nombre `walker` en cada una de las variables.

Código 9.2: Ejemplo de `__init__.py` para el caminador i-Walker

```

1 import Agent
2 import Behaviours
3
4 # Agentes para cada servicio del dispositivo
5 config_agents = { 'walker': Agent.Agent }
6
7 # Comportamientos para cada servicio del dispositivo
8 config_behaviours = { 'walker': Behaviours.Device }

```

9.2 Agente

El agente del caminador tiene el objetivo de procesar los datos provenientes de los sensores del caminador. Este procesado se llevará a cabo **por cada ejercicio** que se realice, y acabará con una lista de variables útiles para el análisis del ejercicio. Como se detalló en la sección 5.1, el caminador genera tres ficheros por cada ejercicio. El archivo más importante para el procesado es el que contiene la extensión `wlk`. Este fichero contiene la lectura de los sensores del caminador cada 100 milisegundos. A través de estos datos y los obtenidos del archivo con extensión `usr`, que incluye la configuración del ejercicio, el agente generará diferentes variables acumulativas durante todo el ejercicio. El archivo restante no se usará en esta implementación, ya que se realizará un procesado parecido al que este contiene. Las variables por cada ejercicio son:

- `nu_l` y `nu_r`: valores del ejercicio fijado para cada una de las ruedas utilizados durante la actividad. Se obtienen de forma directa de la configuración del ejercicio.
- `lambda_l` y `lambda_r`: valores de ayuda para cada rueda. Igual que con el valor anterior, se obtiene del archivo de configuración del ejercicio.
- `time_start`: UNIX *timestamp* en el cual el ejercicio comenzó. Obtenido del archivo `.usr`.
- `exercise`: nombre del tipo de ejercicio. Igual que las variables anteriores, se obtiene del archivo de configuración.
- `time_end`: UNIX *timestamp* del final del ejercicio. Se calcula a través de la resta del último timestamp y el tiempo de inicio. Esto obtiene el tiempo transcurrido en milisegundos. A través de este valor, se divide para obtener el tiempo en segundos y se suma al tiempo inicial, obteniendo así el UNIX *timestamp* final.
- `distance`: distancia total recorrida. Se calcula en cada ciclo haciendo la media aritmética del valor absoluto de las velocidades de ambas ruedas. Este valor se divide por la diferencia de tiempo desde el anterior ciclo y se suma al total.
- `brake_usage`: porcentaje del tiempo en el cual el usuario ha hecho uso de alguno de los frenos de mano. Durante el procesado se cuentan todos los ciclos en el cuál alguno de los frenos está activo. Al final, se divide este valor por el número de ciclos totales.
- `descending`: porcentaje del tiempo que el paciente ha estado bajando una pendiente. Se calcula de una forma similar a la variable anterior.

- **ascending:** porcentaje de tiempo subiendo una pendiente. El cálculo es igual al anterior de forma inversa.

Para poder calcular todos estos valores, hay que tener en cuenta en qué situación está el caminador con respecto al paciente y si está pasando alguna acción reseñable. Para ello, se hace un análisis continuo de los valores para fijar una serie de eventos sucedidos durante el ejercicio. Estos eventos, finalmente, se guardan en la variable **events** para que puedan ser mostradas en el informe del ejercicio. Todos los eventos posibles son:

1. **Se apoya o suelta el caminador:** es el evento más básico. Indica si el paciente está apoyado o no en el caminador. Servirá como base para el resto de eventos. Se determina que el paciente se ha apoyado si la media de la fuerza en el eje *z* durante el último medio segundo (últimos cinco ciclos) es inferior a -1000 cN . De igual forma, se considera que lo ha soltado si ese valor es superior a -1000 cN durante más de medio segundo.
2. **Empieza o para de caminar:** antes de poder determinar si está caminando o no, el paciente debe estar apoyado en el caminador. A partir de ese momento, se considera que empieza a caminar si supera la media de 2 cm/s durante medio segundo. Se establece que ha parado si baja de una media de 1 cm/s , también durante más de medio segundo.
3. **Inicia subida:** si el paciente está caminando y el valor del *tilt* supera los 50° , entonces se considera que ha iniciado la subida de una rampa. De nuevo, se calcula usando la media durante el último medio segundo para evitar errores de medición.
4. **Inicia bajada:** de igual forma que la anterior pero de forma inversa. Se fija si el valor del *tilt* baja de los -50° durante medio segundo o más.
5. **Vuelve a terreno recto:** una vez que el paciente ha empezado a subir o bajar una cuesta, es útil saber cuándo esta ha acabado. Esto se puede calcular guardando una variable que indique si el dispositivo se encuentra en una subida o bajada. Sabiendo esto, sólo hay que ver si el valor del *tilt* vuelve a valores normales ($\pm 50^\circ$) durante medio segundo.
6. **Posibles caída:** a partir de ciertos valores en las inclinaciones puede considerarse que el paciente se ha caído. Esto se calcula, también, mediante las medias de las variables *tilt* y *roll*, cuando superan por más de medio segundos los valores $\pm 400^\circ$.
7. **Caída:** a partir de ciertos valores en la inclinación, se puede estimar sin demasiada duda que el paciente ha caído al suelo. Este valor se ha fijado en

± 800 en alguna de las variables *tilt* y *roll* durante más de medio segundo.

8. **Alerta de fuerza:** el paciente puede tener dificultades en ciertos momentos para realizar algún ejercicio. Esto se verá reflejado en las lecturas a los sensores de fuerza en las manos. En las pruebas se fijaron unos valores de ± 8000 cN para la fuerza vertical (eje *z*) y ± 3000 cN para la fuerza lateral (eje *y*). De nuevo, sucederá cuando se superen esos límites en la media de fuerzas cada medio segundo.

El caminador genera los archivos de los ejercicios en un directorio específico. Para poder procesarlos, por tanto, el agente deberá obtener todos los ejercicios nuevos de forma periódica y procesarlos. Para este fin, dispondrá de una variable de nombre `last_exercise` (persistente en la red social) que guardará el UNIX *timestamp* del último ejercicio analizado. A través de esta variable, cada vez que acceda al directorio donde se guardan los ejercicios, podrá obtener un listado de aquellos que no ha procesado aún. Los analizará y, después, enviará los resultados al agente del paciente, que se encargará de procesarlos. Antes de enviar los datos, sin embargo, **puede que el agente ya haya enviado datos** al paciente. Esto sucede si durante el análisis de alguno de los ejercicios se ha producido una alerta grave. Estos casos incluyen las alertas de caídas y fuerzas. No obstante, esos eventos seguirán apareciendo en la lista enviada al paciente al finalizar el análisis.

9.3 Comportamientos

Una vez el procesado de un ejercicio ha sido enviado al paciente, el agente de este no sabe qué hacer con ellos. Para este fin, la plantilla de comportamientos del caminador añade diferentes comportamientos al paciente. Los agentes de personas relacionadas no han sido modificados con nuevos comportamientos, ya que las funciones básicas de estos son suficientes. No obstante, podrían ser añadidos nuevos comportamientos a estos si fuera necesario. Las nuevas funcionalidades del paciente se encargarán de hacer un proceso de algunos de los datos recibidos en los nuevos ejercicios en comparación con los ejercicios anteriores. Además, gestionarán los diferentes tipos de alertas para poder proceder a avisar a la red social y al resto de agentes relacionados. En las siguientes subsecciones se analiza cada uno de estos puntos de forma más detallada.

9.3.1 Gestión de mensajes

Los comportamientos añaden un `EventBehaviour` para recibir los mensajes directamente del caminador. Este comportamiento es muy sencillo. Su funcionamiento se basa en procesar el mensaje y añadir los conocimientos necesarios dependiendo del contenido del mensaje. A partir de estos conocimientos, el agente realizará las acciones pertinentes si fuera necesario (eso lo estipularán sus reglas y otros conocimientos). Los mensajes que puede obtener el agente son:

- `walker-new-exercises`: contiene un listado de nuevos ejercicios procesados por el caminador. Está codificado en JSON para poder extraer los datos de forma más cómoda. El agente actualizará el conocimiento de la sesión actual, extendiendo el número de ejercicios realizados.
- `walker-fall`: mensaje simple que enviará el caminador cuando detecte una caída del paciente. Se añadirá un conocimiento de peligro (`B(Danger)`), que el agente podrá procesar en el siguiente ciclo.
- `walker-vertical-force-alert`: el mensaje llegará si se detecta un exceso de fuerza vertical por parte del paciente. Al igual que en el caso anterior, se añadirá conocimiento, pero en este caso de aviso (`B(Warning)`).
- `walker-lateral-force-alert`: igual que el elemento anterior, pero para fuerzas laterales.

9.3.2 Gestión de peligros y avisos

La gestión de peligros y avisos se ha implementado usando el modelo BDI diseñado para este proyecto. Este permite fijar un comportamiento (tarea), que se ejecutará cuando se cumplan unas determinadas condiciones y cuando no se cumplan otras. Por tanto, por cada tipo de peligro se establecerá una nueva tarea. Cada uno de las tareas procederán de la siguiente manera:

1. `AlertWalkerFall`: se ejecutará cuando se detecte un peligro de caída. No se repetirá la alerta si ya se ha informado de una caída anterior en los últimos diez minutos.
2. `AlertVerticalForce`: se ejecutará cuando se detecte un aviso de exceso de fuerza vertical, no se este informado ya sobre ello y haya pasado al menos dos horas desde la última alerta.

3. `AlertLateralForce`: igual que el caso de fuerzas verticales pero para la fuerza lateral.

9.3.3 Sesiones

Cada vez que el paciente finalice un ejercicio con el caminador, este será procesado y enviado al agente del paciente. Es en ese momento cuando debe sufrir otro procesado. Uno de los problemas que existe es que estos pueden tener una pequeña duración, realizándose varios durante un periodo de tiempo corto. Para solucionarlo, los nuevos comportamientos del paciente guardarán una variable (enlazada al caminador) con los ejercicios de la sesión actual. La variable tomará el nombre `session`.

Una sesión consistirá en una serie de ejercicios (uno o más) durante un **periodo de tiempo menor a 2 horas** entre ellos. Es decir, una sesión de ejercicios puede perdurar en el tiempo siempre y cuando se realice un ejercicio cada dos horas. Para controlar este aspecto, se ha añadido un comportamiento periódico (`CommitWalkerSession`), que en cada ciclo compara la variable `time_end` del último ejercicio de la sesión actual con el UNIX *timestamp* actual. Si esta diferencia supera las dos horas, se procede a finalizar la sesión y componer un informe de esta.

Además de los datos de la sesión actual, también se guardan algunos datos más de las anteriores sesiones o acumulación de ciertos valores de todas ellas. Estas variables son:

1. `last_fifteen_sessions`: datos procesados de las últimas quince sesiones (como máximo).
2. `reports_sent`: número de informes que se han enviado a la red social con este caminador.
3. `dist_max`: distancia máxima (en metros) que ha realizado durante una misma sesión.
4. `dists_sum`: sumatorio de todas las distancias realizadas durante todas las sesiones con el i-Walker.

9.3.4 Composición de informes

Una vez que el agente detecta que ha finalizado una sesión de ejercicios, este puede proceder a componer un informe, el cual enviará posteriormente a la red social. Antes de poder componer un informe en un lenguaje entendible para los seres humanos,

primero hay que calcular algunos valores importantes para el conjunto de la sesión. Entre estos valores, los dos más importantes son el número de ejercicios, la duración total de la sesión y la distancia recorrida durante todos ejercicios. Partiendo de estos datos base y todos los datos de los ejercicios de la sesión, el informe generado dispondrá de las siguientes secciones:

Resumen

Contiene un pequeño resumen de la sesión. En caso de que la sesión contenga un solo ejercicio, se informará de que el paciente ha realizado un ejercicio y no una sesión. Se informará también si es la primera sesión generada por el caminador y la distancia total que ha realizado esta vez. Además, puede informar sobre un gran uso de los frenos de mano, que pueden indicar que el paciente se encuentra incómodo manejando el caminador, o que no se encuentra cómodo realizando el ejercicio. Por otro lado, el resumen indica también si esta sesión ha sido especial en algún aspecto. En concreto, indicará si la sesión ha marcado un máximo o un mínimo de la distancia recorrida en las últimas sesiones, o también si ha recorrido más distancia que en todas las sesiones anteriores. Por último, si sólo se ha realizado un ejercicio, el resumen contendrá información sobre los valores de ayuda y ejercicio configurados para el ejercicio, valores medios de las fuerzas laterales durante los ejercicios y el porcentaje de tiempo que el usuario ha estado subiendo o bajando pendientes.

Ejercicio

Esta sección sólo se incluirá si la sesión ha contenido un solo ejercicio. Contendrá un listado de eventos útiles producidos durante este. Estos eventos se han detallado en la sección 9.2. Algunos de ellos pueden suceder, por ejemplo, cuando el paciente se apoya en el caminador, empieza a caminar, empieza a subir una pendiente o cuando se detecta una caída. Cada uno de estos eventos vendrá precedido por el tiempo que ha pasado desde que se empezó el ejercicio.

Datos de la sesión

Al contrario que el elemento anterior, esta sección sólo se encontrará en los informes de sesiones con más de un ejercicio. Mostrará los mismos datos incluidos para ejercicios aislados. Para representarlos, se generará una tabla que dispondrá de nueve columnas. Las columnas informarán del tipo de ejercicio, porcentaje de ayuda o ejercicio, media de fuerzas laterales, utilización de frenos y porcentaje de subidas o bajas durante

el ejercicio. Cada ejercicio en la tabla dispondrá de dos filas, la primera de ellas rellenará cada una de las columnas anteriores. La segunda fila ocupará todas las columnas, y contendrá el listado de eventos sucedidos durante el ejercicio.

Tendencia en el tiempo

El agente del paciente guarda una estimación de su estado de ánimo, la cual pueden consultar y modificar todos los dispositivos que este tenga asignados. A través de este valor y los guardados en las sesiones anteriores del caminador, se puede estimar su estado actual y ver si está mejorando o empeorando. El aspecto principal del caminador es, como su nombre indica, caminar. Diversos estudios, como el citado en [45], han comprobado que un aumento en el hábito de caminar puede mejorar el estado de ánimo de personas mayores. Por tanto, a través de los valores de distancia y el de la estimación actual de su estado de ánimo, se puede prever la situación del paciente y si puede mejorar o empeorar más.

El análisis se realizará cuando el paciente supere un número (15) de sesiones con el i-Walker. A partir de este momento, en cada sesión se realizará una **predicción de la tendencia de las distancias** que ha recorrido en estas sesiones. Esta predicción se realiza a través de una *simple linear regression*. La regresión brindará una recta predictiva, a partir de la cual se puede calcular la tendencia en el tiempo mirando su pendiente. Una pendiente positiva grande indicará que el paciente está aumentando la distancia que realiza y, por tanto, indicará una posible mejoría de su estado de ánimo. Por el contrario, una pendiente negativa indicará que está bajando la distancia caminada y, posiblemente, sintiéndose peor. Este valor puede ser usado junto a la estimación de su estado de ánimo actual, produciendo tres predicciones (buen humor, mal humor o estable) por cada rango de pendiente en la regresión. Esto ha producido un total de 27 posibilidades a la hora de calcular una tendencia del paciente. Un ejemplo de los resultados obtenidos puede verse en el capítulo 10.

La predicción se usará también para estimar el nuevo estado de ánimo del paciente. Una pendiente positiva indica que el paciente puede estar mejorando su ánimo, mientras que una pendiente negativa indica puede indicar lo contrario. Los valores de la estimación comprenden el rango de números reales entre -10 y 10. Para estimarlo, en cada sesión se aumentara o disminuirá un máximo de 1 punto, dependiendo de la pendiente de la predicción.

Progreso

Por último, el informe busca informar sobre el progreso que el paciente ha ido realizando a lo largo del tiempo durante las últimas sesiones. Para conseguir esto, la mejor forma es visualizarlo de forma gráfica. Existen multitud de variables que pueden utilizarse para realizar gráficos. No obstante, para no saturar al usuario de datos, se van a realizar gráficos de tres variables fundamentales:

1. Tiempo por sesión
2. Distancia realizada
3. Niveles de ayuda y ejercicio

Estos tres valores permitirán al usuario estimar de un vistazo la mejoría o el empeoramiento del paciente al que estan relacionados.

10

Resultados

Después de implementar todos los sistemas y los servicios de ejemplo, es posible ver los resultados producidos a la hora de usar todos los sistemas al mismo tiempo. Uno de los problemas de este proyecto es que no ofrece datos numéricos, sino que genera contenido y mueve este a través de los diferentes sistemas que se han desarrollado. Para poder ver resultados, pues, hay que probar simular algunas de las diferentes situaciones que podrían darse en la realidad y que pueden ser interesantes para probar los sistemas. En este capítulo se recogerán algunos de estos ejemplos y se mostrarán aquellos los resultados obtenidos.

10.1 Conexión entre sistemas

Uno de los aspectos que más pueden mostrar el resultado de todos los sistemas es la conexión entre la red social y el sistema multi-agente. Esta conexión se realiza a través de la interfaz incluída en cada uno de ellos. La interfaz permite que los sistemas estén conectados para mantener los paralelismos entre usuarios en la red social y los agentes. Esta conexión puede verse fácilmente creando nuevos elementos o relaciones en la red social y viendo los cambios que se producen en el sistema multi-agente.

En el siguiente ejemplo, la red social cuenta con un único usuario (además del administrador) con el rol de médico. Esto significa que el sistema multi-agente estará vacío al no existir ningún paciente y, por tanto, ningún grupo paciente-relaciones. Para que el sistema multi-agente empiece a trabajar completamente, será necesario añadir un paciente.

```
Importando configuracion de dispositivos
Bottle v0.11.6 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.
```

```
0 grupo/s recibidos...
manager@127.0.0.1 => (Manager) Iniciando
Sistema iniciado, Ctrl+C para parar los agentes
```

Las líneas anteriores muestran la salida estándar al iniciar el sistema multi-agente. Se puede apreciar como ha usado la interfaz entre sistemas para obtener todos los grupos de pacientes. Al no existir pacientes, no ha conseguido obtener ningún grupo. Por tanto, el único agente que ha iniciado su ejecución es el agente gestor (**Manager**). El siguiente paso es crear un nuevo paciente, como se puede ver en las imágenes de la figura 10.1. La primera muestra el paso de creación del usuario y la segunda el perfil de usuario creado, que ha generado un identificador pa70352f4 en la red a través de su identificador real.



Figura 10.1: Pasos para la creación de un paciente en la red social

Finalmente, puede apreciarse como el sistema multi-agente recibe una petición para añadir el nuevo usuario creado con el rol de paciente. El sistema crea el agente con nombre pa70352f4 y lo inicia. Después, responde a la interfaz indicando que no ha habido errores. Las siguientes líneas muestran la salida estándar del sistema multi-agente atentiendo esta petición.

```
Request
username => pa70352f4
patientGUID => 442
key => lml41CdmhzE3nP8ymdIZ504kzaUiNCygecN
guid => 442
type => PATIENT_ROLE
method => person.add

pa70352f4@127.0.0.1 => (Paciente) iniciando
{'status': 0, 'message': 'ok'}
```

La red social ya contaba con un médico, pero este no se ve representado por ningún agente. Esto es debido a que en este proyecto el sistema multi-agente se basa en los

grupos de pacientes, sus dispositivos y sus relaciones. Este aspecto se decidió así ya que el procesado de los datos de los servicios asistenciales era lo más importante. Por tanto, como el médico no está relacionado con ningún paciente y, por tanto, con ningún dispositivo activo, no se inicia ningún agente para él. Sin embargo, en un futuro el agente de un médico podría tener más rutinas o servicios independientemente de los dispositivos. Cambiar este aspecto sería fácil, sólo habría que modificar el método de la interfaz que obtiene los grupos de la red al iniciar el sistema y eliminar la restricción que añade agentes sólo a los pacientes al crear un nuevo usuario en la red (los demás se añaden al añadir relación con un paciente). Por tanto, para que el agente del médico se cree hay que relacionar a este médico al nuevo paciente. La figura 10.2 muestra cómo se añade esta relación entre el médico y el nuevo paciente añadido anteriormente.



Figura 10.2: Interfaz para añadir relaciones a un usuario

Esta nueva relación hará que la red social envíe una petición al sistema multi-agente. A su vez, esto hará que el gestor de agentes inicie un nuevo agente para el doctor. Además, este se añadirá al grupo que se creó anteriormente. A partir de ese momento, entonces, los agentes de los dos usuarios de la red estarán también relacionados. Las siguientes líneas muestran la petición atendida y el inicio del nuevo agente del doctor.

```
Request
username => doa88a014
patientGUID => 442
key => lml41CdmhzE3nP8ymdIZ504kzaUiNCygecN
guid => 443
type => DOCTOR_ROLE
method => person.add
doa88a014@127.0.0.1 => (Doctor) iniciando
{'status': 0, 'message': 'ok'}
```

El siguiente paso para ver los resultados de la conexión entre los sistemas es añadir dispositivos al paciente. En este caso, el dispositivo será el i-Walker, dispositivo del que se han desarrollado servicios de ejemplo. Las imágenes de la figura 10.3

muestran el resultado de este paso en la red social. En la primera de ellas aparece el cuestionario usado para añadir un nuevo dispositivo. Por otro lado, en la segunda imagen aparece el formulario para asignar un nuevo dispositivo al paciente.

Figura 10.3: Creación y asignación de un dispositivo en la red social

El sistema multi-agente recibirá una nueva petición para añadir el dispositivo. La petición se procesará añadiendo un nuevo dispositivo y asignándolo al grupo del paciente. El dispositivo se añadirá, pero se mantendrá como inactivo hasta que el agente de este se cree en su propia máquina y se comunique con el agente gestor. Las siguientes líneas muestran la salida del sistema multi-agente atendiendo esta petición. Se puede ver que, efectivamente, no crea ningún agente como pasó al añadir el nuevo paciente.

```
Request
patientName => pa70352f4
patientGUID => 442
deviceID => wlk-42
key => lml141CdmhzE3nP8ymdIZ504kzaUiNCygecN
guid => 451
type => walker
method => devices.add
{'status': 0, 'message': 'ok'}
```

Para acabar de completar este paso, por tanto, se debe iniciar el agente del i-Walker desde la misma máquina del dispositivo. A continuación puede verse la salida del agente en el caminador. Primero, se observa como el agente se configura consultando los datos del paciente con el gestor (detailed en la sección 7.3.4). Después, busca nuevos ejercicios y los analiza.

```
Importando configuracion de dispositivos
wlk-42@127.0.0.1 => Configurando
wlk-42@127.0.0.1 => Esperando respuesta del manager
wlk-42@127.0.0.1 => (Walker) iniciando
```

```
wlk-42@127.0.0.1 => Buscando ejercicio mas nuevo
wlk-42@127.0.0.1 => Analizando nuevo ejercicio
...
```

Por último, la conexión entre sistemas también permite eliminar elementos de los sistemas. Uno de estos ejemplos puede darse al desasignar el i-Walker que se había añadido al paciente. En la imagen de la figura 10.4 se aprecia el paso a seguir para conseguir esto en la red social.

Identificador	Tipo	Opciones
wlk-42	i-walker	Desasignar

Figura 10.4: Proceso de desasignación de un dispositivo a un paciente en la red social

En este caso, el dispositivo no se eliminará de la red social, sino que sólo se eliminará la relación con el paciente. Sin embargo, en el sistema multi-agente el gestor comunica la desasignación al agente del dispositivo. Este último finaliza su comportamiento, ya que, al no estar asignado, no captará datos y no podrá comunicarse con ningún paciente. En caso de asignarse otra vez, el agente se volvería a configurar automáticamente (relacionándose con el nuevo paciente) cuando se iniciara de nuevo. Esta parte del diseño se explicó en la sección 7.3.4.

```
Request
guid => 451
method => devices.delete
key => lml41CdmhzE3nP8ymdIZ504kzaUiNCygecN
{'status': 0, 'message': 'ok'}
manager@127.0.0.1 => Informando desconexion a dispositivo
# TTY del dispositivo
wlk-42@127.0.0.1 => Parando agente por peticion del gestor
```

En resumen, en estos resultados puede verse como la conexión entre los sistemas es satisfactoria. La conexión mantiene los paralelismos diseñados para ambos sistemas y ambos sistemas se encargan de que no se produzcan inconsistencias usando esta interfaz. El trabajo que resta en este punto es procesar los datos de los dispositivos en el sistema y mandarlos de vuelta a la red social.

10.2 Informes de sesión

El sistema multi-agente puede producir informes sobre el estado del paciente. En este proyecto se han incluído la composición de informes para sesiones con el caminador i-Walker. Los comportamientos del dispositivo esperan hasta que finalice una sesión de ejercicios. Después, hacen un análisis de diferentes aspectos importantes en estos datos y generan un informe que envían a la red social. Estos informes siguen las características descritas en la sección 9.3.4. Para ver los resultados de esto, se van a usar algunas de las pruebas reales que se hicieron con el caminador.

El primer ejemplo consiste en una sesión de un solo ejercicio. El ejercicio es una de las pruebas realizadas con el i-Walker descritas en la sección 5.2. Consistió en recorrer un pasillo girando en ambas direcciones continuamente. La figura 10.5 muestra el informe realizado para esta actividad. Puede apreciarse como el resumen indica que se ha hecho un solo ejercicio y de qué tipo era., la distancia que se ha recorrido, etc. También se muestra la lista de eventos ocurridos durante el ejercicio y la tendencia del paciente durante las últimas sesiones. Finalmente, se observa como el informe incluye un gráfico de la duración de las últimas sesiones. Además de este gráfico, también se incluyen dos más que muestran la distancia y los porcentajes de ayuda y ejercicio de las mismas sesiones anteriores. Estos no se muestra en la imagen ya que no añade ningún aspecto importante más y ocuparían mucho espacio.

El otro ejemplo incluye con una sesión con tres ejercicios diferentes. Estos ejercicios incluyen el visto anteriormente y dos ejercicios parecidos. En estos dos nuevos ejercicios se configuró el caminador con la máxima ayuda posible, y después con el máximo nivel de ejercicio. El informe resultante para esta sesión puede verse en la figura 10.6. Puede observarse como el resumen es menor e incluye datos muy generales. Después, el informe muestra una tabla con los datos importantes para cada ejercicio y la lista de eventos producidos. Este informe también incluye los gráficos mencionados anteriormente y la estimación de la tendencia en el tiempo, que no se han incluído.

10.3 Notificaciones de peligro

Un aspecto importante entre la red social y los dispositivos es la gestión de notificaciones. Estas notificaciones pueden producirse ante eventos importantes, tales como alertas o avisos producidos directamente por los dispositivos o durante el procesado de sus datos. Para ver los resultados producidos en este aspecto, se contará con una

Resumen

Ha realizado un ejercicio de Stretching con el caminador. Esta vez ha caminado un total de 15 metros en 47 segundos. El caminador fue configurado con un valor de ayuda de 0/100 y un valor de ejercicio de 0/100. Con esta configuración, el paciente empujó el caminador con una fuerza media de 2.2 N

Ejercicio

- 3s** Se apoya en el caminador
- 3s** Empieza a caminar
- 45s** Para de caminar (15 metros)
- 45s** Suelta el caminador

Tendencia en el tiempo

Los últimos ejercicios realizados muestran que el paciente ha sufrido una bajada brusca de la distancia caminada. Si no está siendo atendido ya, necesitaría una revisión para comprobar su estado.

Progreso

Tiempo por sesión

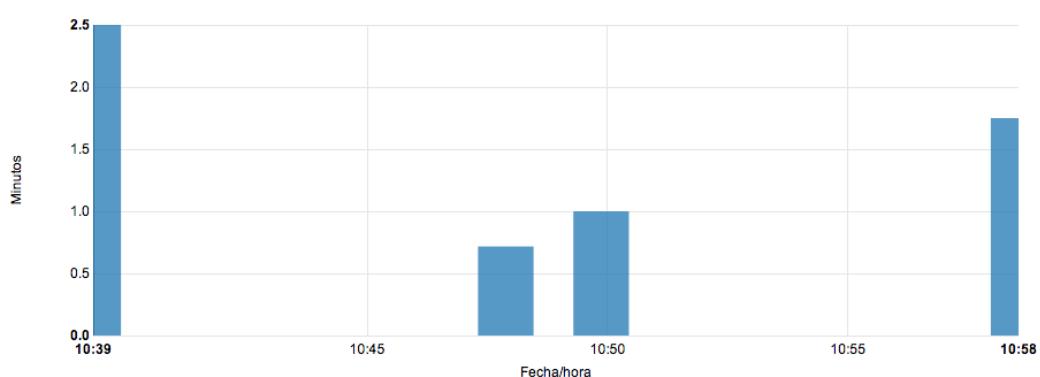


Figura 10.5: Ejemplo de informe de un ejercicio generado por el i-Walker

Resumen

Ha finalizado una sesión de ejercicios usando el caminador. Esta vez ha caminado un total de 43 metros en 2.5 minutos.

Datos de la sesión (3 ejercicios)

Ejercicio	Tiempo	Ayuda	Ejercicio	Empuje	Distancia	Frenos	Subida	Bajada
Stretching	47 segundos	0/100	0/100	2.2 N	15 metros	0/100	0/100	0/100
3s Se apoya en el caminador 3s Empieza a caminar 45s Para de caminar (15 metros) 45s Suelta el caminador								
Stretching	43 segundos	100/100	0/100	2.2 N	15 metros	0/100	0/100	0/100
2s Se apoya en el caminador 2s Empieza a caminar 41s Para de caminar (15 metros) 41s Suelta el caminador								
Stretching	60 segundos	0/100	100/100	8.2 N	13 metros	0/100	0/100	0/100
2s Se apoya en el caminador 2s Empieza a caminar 59s Suelta el caminador								

Figura 10.6: Ejemplo de informe de una sesión generada por el i-Walker

red social con un único paciente y un caminador i-Walker. El paciente estará relacionado con otro usuario, que representará un doctor que le ofrece tratamiento. En la siguiente imagen puede verse la salida en el sistema multi-agente del dispositivo. Se ha incluído un ejercicio en el cual se simula una caída. Se observa como el agente del dispositivo analiza el ejercicio y detecta la caída. Después, procede a informar sobre ello al paciente. Además, también envía el procesado de todo el ejercicio.

```
wlk-42@127.0.0.1 => Configurando
wlk-42@127.0.0.1 => Esperando respuesta del manager
wlk-42@127.0.0.1 => (Walker) iniciando
wlk-42@127.0.0.1 => Buscando ejercicio mas nuevo
wlk-42@127.0.0.1 => Analizando nuevo ejercicio
wlk-42@127.0.0.1 => Caida detectada
wlk-42@127.0.0.1 => (Device) Enviando mensaje a paciente
```

En el otro lado del sistema multi-agente (servidor), el paciente recibirá primero el aviso de caída detectado por el i-Walker. Añadirá este conocimiento a su base de conocimiento. El conocimiento actual del entorno le permitirá notificar esta caída, por lo que finalmente informará sobre ella a todos los agentes relacionados. El agente del doctor, el único agente relacionado con el anterior, recibirá esta información y la propagará a la red social. Todos estos aspectos pueden verse en las siguientes líneas, que muestran la salida estándar del sistema.

```
pa70352f4@127.0.0.1 => +b B(Danger,451,Walker_fall,1388344793)
pa70352f4@127.0.0.1 => Generando informe de sesion
pa70352f4@127.0.0.1 => Informando sobre caida en i-Walker
doa88a014@127.0.0.1 => Enviando notificacion de alerta a la red
```

Por último, la red social del doctor mostrará una nueva notificación informando sobre esta caída. Esta notificación aparecerá de forma instantánea si el usuario tiene abierta la red social. En caso contrario, podrá verla cuando inicie sesión. La imagen en la figura 10.7 muestra la notificación específica para el ejemplo de la caída con el caminador.

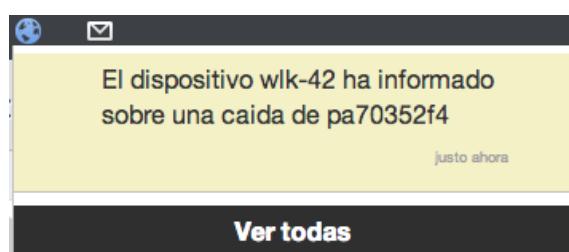


Figura 10.7: Ejemplo de notificación en la red social

11

Conclusiones y trabajo futuro

A medida que se avanza en el documento, pueden leerse diferentes conclusiones a las que se ha llegado en las diferentes partes del desarrollo. Estas conclusiones incluyen trabajo que no se ha llegado a realizar pero que pueden ser buenas ideas para proyectos futuros. En este capítulo se resumen todas estas conclusiones, y se realiza una conclusión final del global del proyecto.

Como ha sucedido hasta ahora, se van a separar las diferentes conclusiones en las cuatro grandes secciones del desarrollo. En primer lugar la red social, después el sistema multi-agente, la interfaz entre ambos y finalmente el desarrollo de servicios asistenciales de ejemplo.

11.1 Red social

Las redes sociales tienen un **rango de complejidad muy elevado**. En su forma más básica, una red social se puede representar como un grafo $G = (V, E)$, donde sus vértices (personas) pueden relacionarse (aristas) con otros vértices de alguna manera. Una aplicación que trabaje con una red social, pues, podrá tener múltiples niveles de complejidad dependiendo de la cantidad de características que se quieran ofrecer. En este proyecto se ha implementado una red social de complejidad media. Además de relacionar personas, esta ofrece comunicación básica entre ellas, diferentes tipos de roles y otros vértices no personales (dispositivos, mensajes). Este aspecto, pues, deja **mucho margen** a la hora de desarrollar nuevas funcionalidades o ampliar las que ya han sido implementadas. Además, también pueden surgir algunos problemas. Más en detalle:

1. **Roles:** la cantidad de tipos de usuarios implementados es muy limitado. Los roles desarrollados incluyen pacientes, familiares, doctores, enfermeros y administradores. Los roles, por tanto, incluyen solo el círculo personal de los pacientes y roles administrativos. En futuros proyectos se podrían desarrollar más roles dentro del ambiente médico y nuevas características relacionadas con estos roles. Por ejemplo, podrían desarrollarse roles relacionados con la farmacia. De esta forma, un paciente podría relacionarse con su farmacéutico para que este llevara un control sobre los medicamentos que debe tomar y

cuándo el paciente debe obtener más. Sin embargo, esto generaría un nuevo nivel de complejidad, cuestiones de privacidad, etc.

2. **Nuevas funcionalidades:** ya se ha comentado que una red social puede tener una cantidad muy grande de funcionalidades. En el caso de esta, podrían incluirse nuevas que ayudaran al tratamiento del paciente o a su autoestima. Por ejemplo, podrían incluirse juegos didácticos, o también vídeos que promovieran la vida sana y activa. En resumen, existen infinidad de nuevas funcionalidades que podría albergar la red en relación con el entorno médico en el que está anclada.
3. **Ampliación de funcionalidades:** además de añadir nuevos módulos, también pueden ampliarse los ya existentes. Todos los módulos de la red social se han implementado de la forma más modular posible. Esto significa que cada módulo tiene una función muy determinada. Esto hace que algunos de ellos sean realmente simples. De esta manera, se deja la puerta abierta a futuros desarrolladores para aumentar la complejidad de estos módulos añadiendo más funcionalidades. Un ejemplo de esto puede verse en la página de actividad. Esta muestra información sobre los usuarios relacionados, pero podría personalizarse más para que la información cambiara dependiendo del rol. También sucede esto con los perfiles de los usuarios o las páginas de cada dispositivo, que podrían albergar mucha más información de diferentes formas.
4. **Privacidad:** en la implementación de la red se ha decidido dejar a un lado las cuestiones complejas sobre privacidad. En este momento, las páginas de los usuarios sólo pueden ser accedidas por los propios usuarios, por los administradores, o por aquellas personas que guarden alguna relación con ellos. Además, los usuarios se identifican con un número pseudo-aleatorio generado a partir de su identificador real (DNI, por ejemplo). Sin embargo, existen muchas más cuestiones sobre privacidad que deberían ser tenidas en cuenta dependiendo del país donde los sistemas se lleven a producción. Es decir, las leyes podrían ser diferentes dependiendo del sitio donde se usen los sistemas. Por tanto, es un aspecto que queda abierto para futuros proyectos, que podrían implementar las cuestiones de privacidad de diferentes maneras. No obstante, en la sección 12.5.4 se detallan estas normativas para el caso particular de España y la Unión Europea.
5. **Escalabilidad:** la naturaleza del proyecto ha llevado a escoger el motor de redes sociales más dinámico para facilitar la implementación. Este aspecto, aunque es de gran ayuda para el proyecto, puede causar problemas a la hora que el número de usuarios aumente de forma rápida. Si sucede, es posible que el rendimiento disminuya o que los requisitos de *hardware* para poder mantener

la red sean muy elevados. En este caso, la solución sería una implementación bajo un motor con un modelo de datos más estricto.

11.2 Sistema multi-agente

Al igual que en la red social, el sistema multi-agente puede recibir mejoras a sus funcionalidades. No obstante, en este caso la arquitectura es más estricta y no ofrece tantas posibilidades. En concreto, a continuación se describen algunos de estos aspectos que se han visto en el transcurso del documento:

1. **Más tipos de agentes:** en el proyecto se incluyen unos pocos tipos de agente relacionados con los roles en la red social, pero podrían incluirse nuevos. Estos nuevos tipos de agentes interactuarían con los ya existentes, que tendrían que ser modificados en mayor o menor medida. Además de nuevos roles personales, también podría ser posible implementar nuevos agentes no personales que gestionaran ciertos aspectos. Un ejemplo de esto podría ser un agente que se encargara de gestionar el transporte (ambulancia, transporte especial, etc.) de pacientes con grandes impedimentos físicos.
2. **Modelo de agentes:** la plataforma de agentes utilizada (SPADE) ofrecía ya una implementación del modelo de agentes que se decidió usar (BDI). No obstante, por diversos problemas se decidió implementarlo de nuevo para así adecuarlo mejor a las características del proyecto. Esta implementación dejó de lado algunas características más complejas que sí estaban presentes en SPADE. Por otro lado, incluyó otras que no lo estaban. Una de estas características que no se han incluido en el proyecto es la composición de planes dinámica a través de diferentes servicios. Este aspecto no es necesario para el proyecto en su estado actual, pero podría ser algo muy útil a la hora de ampliarlo.
3. **Gestor:** el agente encargado de gestionar los grupos de los agentes personales y dispositivos es una pieza clave en la arquitectura del sistema. Por tanto, es un agente que puede sufrir modificaciones a la hora de añadir nuevas funcionalidades en el sistema o modificar las actuales. En su mayoría, estas modificaciones tratarían sobre la comunicación con los posibles nuevos tipos de agente introducidos. Por ejemplo, un nuevo tipo de agente podría necesitar cierta información específica sobre un grupo dentro del sistema, o sobre un dispositivo. Esto haría necesario la modificación directa del agente, o quizás influiría en el desarrollo de una API para que los desarrolladores pudieran implementar cualquier comportamiento dentro del gestor.

4. **Paciente:** uno de los aspectos más complejos a decidir a la hora de diseñar el sistema fue qué información guarda un paciente. Esto se debe a que cada dispositivo dispondrá de información específica recogida en sus propios sensores. El problema surge cuando hay que juntar estos datos. En este proyecto se decidió que el paciente sólo guardaría una estimación de su estado de ánimo y una lista de necesidades y posibles peligros. A través de esta información, los comportamientos de los dispositivos podrían decidir cómo procesar su información. Cada dispositivo puede generar datos de todo tipo y de diferente complejidad. Por tanto, este aspecto necesita aún de mucho estudio, ¿cómo se puede procesar eficientemente la información de varios dispositivos asistenciales para deliberar sobre el estado de un paciente?

11.3 Interfaz entre sistemas

La interfaz entre la red social y el sistema multi-agente se ha implementado como dos servidores, uno en cada sistema. Estos servidores recogen las peticiones del otro y procesan los datos necesarios en su sistema correspondiente. Por tanto, la interfaz sigue un diseño particular pero no está limitada. Esto implica que puede ampliarse de cualquier forma dependiendo de las necesidades que se generen en futuros proyectos.

Otro aspecto que podría tenerse en cuenta en la interfaz de la red es abrir ciertas funcionalidades a terceros desarrolladores. Es decir, ofrecer información limitada a desarrolladores externos para que implementen sus propias aplicaciones. Esto podría permitir realizar aplicaciones para diferentes sistemas operativos móviles, algo que podría ser útil, incluso para llevar la red incluida en alguno de los dispositivos asistenciales más complejos (los que pueden incorporar una *tablet*, por ejemplo). No obstante, no sería uno de los proyectos más interesantes a realizar en un futuro.

11.4 Dispositivos asistenciales

Sólo se ha implementado servicios para un dispositivo, el caminador i-Walker. Por tanto, el trabajo futuro en este aspecto es claro, desarrollar nuevos servicios para dispositivos asistenciales. La arquitectura de los sistemas se ha diseñado pensando en este aspecto. Por ejemplo, en el sistema multi-agente hay plantillas sobre las cuales implementar las clases y comportamientos de nuevos dispositivos. Además, la red social también ofrece opciones para introducir nuevos tipos de dispositivos y, después, crear dispositivos de ese tipo.

Las opciones de nuevos dispositivos son ilimitadas. La arquitectura sólo fija que un dispositivo se compone de un agente y una lista de comportamientos a añadir a las personas relacionadas con el paciente. El agente y los comportamientos pueden tener una complejidad indefinida. Por ejemplo, el agente simplemente podría captar los datos de los sensores del dispositivo y enviarlo al agente del paciente. Los comportamientos añadidos a este podrían obtener estos datos y enviarlos directamente a la red. En el caso contrario, el dispositivo podría contar un ecosistema completo de agentes, que procesarían los datos como fuera necesario. Al acabar este proceso, no obstante, el mismo agente que en el caso anterior tendría que enviar los datos al paciente. A este lado, de igual manera, los comportamientos añadidos podrían tener una gran complejidad. En resumen, las posibilidades a la hora de añadir dispositivos son muy grandes y podrían generar multitud de nuevos proyectos.

11.5 Conclusiones finales

El proyecto ha logrado **alcanzar todos los objetivos** que se fijaron en un inicio. Este ofrece una herramienta para el desarrollo de servicios asistenciales usando un sistema multi-agente integrado en una red social médica. Para ello, se ha realizado un gran trabajo sobre redes sociales, sistemas multi-agente, dispositivos asistenciales y la conexión entre todos ellos. La características de cada uno de estos campos, no obstante, hacen que el número de funcionalidades que pueden crearse sea enorme. Por esa razón, el desarrollo se haya tenido que limitar en diferentes secciones para finalizar el proyecto en el tiempo establecido. Este aspecto ya se tuvo en cuenta desde un inicio, donde se fijaron unos límites y exclusiones en los objetivos para que ninguna de sus partes tomaran demasiado tiempo en ser desarrolladas.

Las características del proyecto, por tanto, han llevado a un diseño e implementación lo más **simple y dinámico** posible. De esta manera, los futuros desarrolladores podrán añadir, modificar o eliminar funcionalidades fácilmente de diferentes maneras. En este aspecto, sin embargo, existen dos partes que ofrecen mayores posibilidades. Por un lado, la red social puede incluir multitud de nuevas funcionalidades. Por otro lado, se pueden desarrollar infinidad de nuevos dispositivos asistenciales que los pacientes podrían utilizar de diversas formas.

En resumen, el proyecto ha cumplido satisfactoriamente sus objetivos. No obstante, **puede servir como origen** para otros nuevos desarrollos ya que existe mucho trabajo que aún podría realizarse en futuros proyectos.

12

Gestión del proyecto

Antes y durante el desarrollo técnico del proyecto se han llevado a cabo diferentes tipos de gestiones para asegurar su finalización. Estas gestiones incluyen planificación del tiempo, presupuesto, metodologías de trabajo, impacto del proyecto en diferentes aspectos y, por último, regulaciones y normativas que le afectan. En las siguientes secciones se detalla cómo se diseñaron cada una de estas gestiones antes de iniciar el desarrollo técnico, y si este ha provocado algún tipo de cambio importante.

12.1 Planificación

12.1.1 Planificación original

Antes de iniciar la planificación inicial, se estudiaron las fechas clave en el proyecto. Este empezaba en julio de 2013 (planificación del proyecto) y finalizaba en enero de 2014. Se fijó que la mayoría del trabajo debería estar finalizado para diciembre, para así tener más tiempo para finalizar la memoria y preparar la defensa. Se fijó, no obstante, septiembre como fecha inicial de la parte técnica del proyecto.

Una vez se conocieron las fechas límite del proyecto, se decidió enumerar las diferentes etapas del proyecto. Estas etapas estaban separadas en dos grupos. El primero de ellos contenía las tareas relacionadas con el desarrollo y el segundo las relacionadas con la preparación de la memoria y la defensa. Las tareas de desarrollo eran mayores en número y, además, contenían una mayor dependencia entre ellas. Estas tareas podían separarse en cuatro etapas principales: **especificación, diseño, implementación y pruebas**.

Las tareas del segundo grupo eran distintas. Por una parte, la mayor parte de la memoria podía escribirse a medida que avanza el desarrollo. Por otra parte, la preparación de la defensa deberá hacerse sólo cuando el resto de trabajo esté acabándose. Por estas razones, la programación del tiempo estaría enfocada en mayor parte a las tareas de desarrollo.

El siguiente paso fue fijar todos los recursos que se iban a utilizar durante el proyecto. Era importante saber qué recursos iban a ser usados en cada fase del proyecto para

poder predecir costes y posibles problemas. Al tratarse de un proyecto de *software* desarrollado por una única persona y su director, el número de recursos a utilizar era bajo. Todos los recursos que se especificaron fueron:

- *Software* para creación de esquemas y diagramas. Utilizado durante la especificación y diseño de los sistemas.
- Editor de código o IDE. Utilizado principalmente durante la implementación.
- Recursos humanos (desarrollador y director). Presentes durante todo el proyecto.
- Documentación. Utilizada durante el diseño y la implementación, en especial a la hora de escoger los sistemas a utilizar (motor de redes sociales y sistema de agentes).
- Servidores Web. Utilizados durante la implementación y la fase de pruebas.
- Ordenador personal. Utilizado durante toda la realización del proyecto.
- Acceso al caminador robótico i-Walker. Se necesitará acceso real al caminador durante la implementación y la fase de pruebas.

A continuación se fijaron las diferentes tareas en cada una de las etapas y sus niveles de riesgo, que indicaba el efecto general que podría tener un retraso o error sobre el resto del proyecto o el uso de recursos. Solo se incluían las tareas relacionadas directamente con el desarrollo, debido a la relación directa entre ellas y la escritura de la memoria y defensa. Las tareas especificadas fueron:

1. Especificación
 - a) Especificar el sistema multi-agente. [riesgo alto]
 - b) Especificar las características de la red social. [riesgo medio]
 - c) Especificar las características de la interfaz de la red social. [riesgo alto]
 - d) Especificar los servicios para el caminador i-Walker. [riesgo medio]
2. Elección de sistemas y diseño
 - a) Elección del sistema multi-agente. [riesgo medio]
 - b) Diseño de los agentes dependiendo del sistema escogido. [riesgo alto]
 - c) Diseño de la interacción del sistema multi-agente con la red social. [riesgo medio]
 - d) Elección del motor de redes sociales. [riesgo alto]

- e) Elección del tipo de servidores a utilizar. [riesgo bajo]
- f) Elección del *software* para el servidor Web. [riesgo bajo]
- g) Diseño de la distribución de datos y agentes. [riesgo alto]
- h) Diseño de la arquitectura (módulos e interacción). [riesgo medio]
- i) Diseño de estructuras de datos complejas. [riesgo medio]
- j) Diseño de algoritmos. [riesgo medio]

3. Implementación

- a) Adaptación del motor de redes sociales para la integración de la interfaz. [riesgo alto]
- b) Implementación de la interfaz de la red. [riesgo alto]
- c) Implementación del sistema multi-agente. [riesgo medio]
- d) Implementación de los servicios de ejemplo. [riesgo medio]

4. Pruebas y corrección de errores

- a) Pruebas con el caminador i-Walker. [riesgo alto]
- b) Realización de tests unitarios. [riesgo bajo]
- c) Corrección de errores. [riesgo medio]

El siguiente paso fue realizar una programación del tiempo para las tareas anteriores y, además, fijar alternativas en caso de retrasos. La figura 12.1 muestra un diagrama de Gantt con la planificación temporal original. Cada tarea se fijó con un tiempo estimado al alza (más tiempo del esperado). No se incluyó la redacción completa de la memoria, ya que se pensaba realizar a medida que avanzaba cada parte del desarrollo y, por tanto, no disponía de precedencias precisas. Las tareas de color rojizo representaban el camino crítico, es decir, tareas que no podían sufrir retrasos ni adelantos. Por otra parte, en color azulado estaban representadas aquellas que permiten modificaciones en su calendario. Las tareas no críticas empezaban tan pronto como fuera posible (MIC¹), pero podrían retrasarse hasta que su finalización empezara a solaparse con una tarea de la que son predecesoras (MAC²).

Finalmente, se establecieron planes de acción para solventar posibles problemas u obstáculos. La planificación se realizó de tal forma que permitiera formular planes alternativas para posibles retrasos en determinados grupos de tareas sin dependencias

¹Fecha mínima de comienzo.

²Fecha máxima de comienzo.

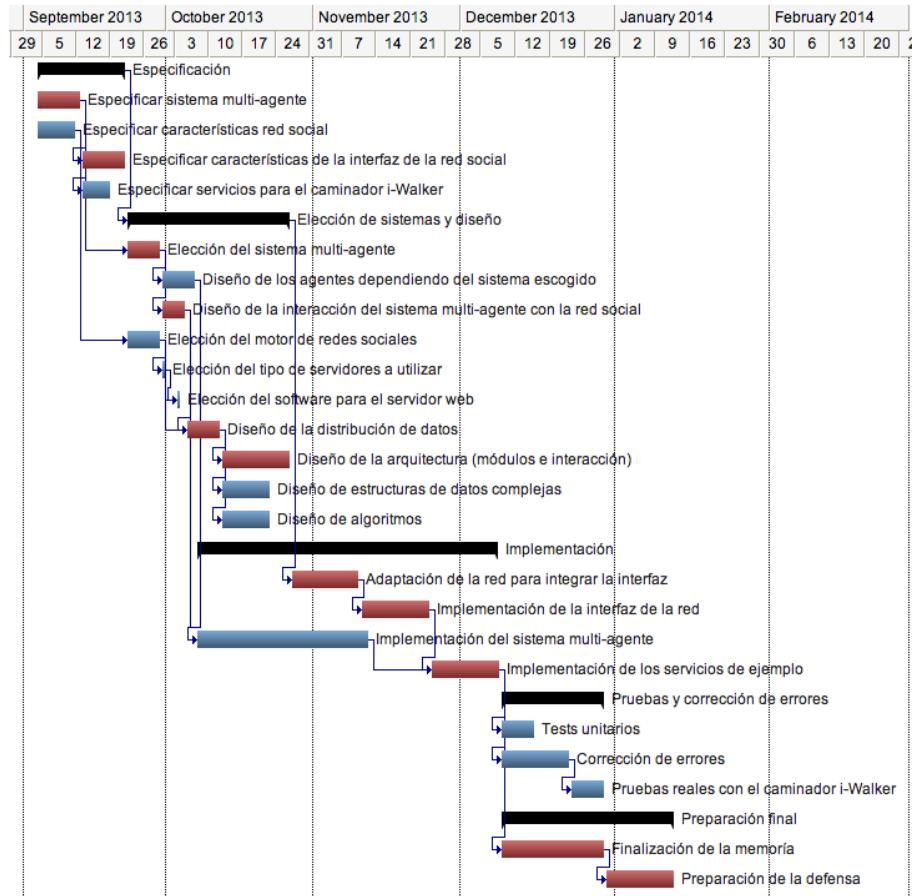


Figura 12.1: Diagrama de Gantt de la planificación temporal original

entre ellas. Teniendo esto en cuenta, se definieron diferentes planes de acción para posibles problemas. Estos planes fueron:

1. Incompatibilidades entre sistemas y lenguajes de programación. La única incompatibilidad que podía darse entre sistemas es la conexión del sistema multi-agente con la interfaz de la red social. Debido a que esta conexión no se realiza hasta muy tarde y los sistemas se escogen durante el diseño, hay tiempo suficiente para adaptarlos. En caso de no poder solucionarlo, se procedería a disminuir los requisitos iniciales para poder dedicar más tiempo a esa tarea durante la implementación. El uso de los recursos no se verá afectado, ya que sólo se modificaría el tiempo de las tareas durante la implementación.
2. Falta de documentación del *software* utilizado. En caso de que la implementación fuera muy lenta por falta de documentación, el plan de acción a seguir consistiría en aumentar su duración, disminuyendo el tiempo dedicado a las pruebas.

Debido a eso, se vería reducido el uso del servidor y el caminador i-Walker, que se usarían en mayor medida durante la fase de pruebas.

3. Especificaciones o diseños pobres. Si la especificación o el diseño no superan la validación, habría que modificar en mayor o menor medida sus contenidos. El plan de acción en este caso consistiría en dedicar menos tiempo a la fase de pruebas, extendiendo la fase de implementación. En un caso extremo, se procedería a reducir los requisitos del proyecto para garantizar así su finalización. En cualquier caso, el uso de recursos materiales no se vería afectado al no modificar la suma total de tiempo entre implementación y pruebas.
4. Problemas de eficiencia algorítmica. En caso de producirse problemas de eficiencia algorítmica, se intentarían solventar respetando el calendario acordado. Si no es posible solucionar el problema, se implementaría una versión menos eficiente. Por otro lado, el uso de recursos no se vería afectado al no modificarse en ningún caso los tiempos planificados.
5. Falta de literatura en la integración de un sistema multi-agente y una red social. Este problema ya se tuvo tenido en cuenta a la hora de realizar la planificación, que incluye el tiempo necesario para su especificación y diseño. No obstante, la falta de literatura podría provocar una especificación o diseño pobres.
6. Fallos de programación en las diferentes implementaciones. Todo sistema tiene ciertos errores de programación. Para dar por finalizado el proyecto, sin embargo, los más importantes tienen que estar solucionados. El plan de acción para solucionar en este caso era utilizar un sistema de monitorización de errores y, al detectar uno, clasificarlo por su nivel de importancia. Durante la fase de implementación y pruebas se deberían ir solucionando por orden de importancia. En caso de no poder solucionarlos todos, se dedicaría menos tiempo a las pruebas y a la preparación de la memoria y la defensa.

Además de estos problemas específicos, la disposición de las tareas hacía que algunas pudieran empezar antes o después. Estas tareas eran:

- El desarrollo del sistema multi-agente y de la red social se mantiene relativamente separado hasta la unión de ambos a la hora de implementar los servicios de ejemplo. Por tanto, un retraso en alguno de ellos podría ser fácilmente compensado con el otro sin que la duración del proyecto ni el uso de recursos sufrieran modificaciones.
- En las últimas etapas del proyecto, las tareas relacionadas con las pruebas y la preparación de la memoria y la defensa se solapan. Si se diese el caso de un retraso, se podría dedicar menos tiempo a alguna de ellas. En cualquier caso, cambiaría el uso de los recursos materiales (servidores y acceso al caminador

i-Walker).

- Un retraso en el diseño podría ser compensado con un inicio posterior de la implementación del sistema multi-agente, que ofrece una dinámica mayor en cuanto al calendario. Si el retraso no es muy grande, la duración total del proyecto no debería verse afectada.
- Se disponeía de un pequeño margen de tiempo en determinadas tareas durante el diseño, que podría ser utilizado para compensar retrasos en otras tareas más importantes.
- Retrasos grandes en la especificación o el diseño provocarían un retraso general en el proyecto. Este retraso se debería solventar con una implementación más pobre o la reducción del tiempo dedicada a las pruebas y corrección de errores. De nuevo, el uso de los diferentes recursos materiales se vería afectado.

12.1.2 Cambios en la planificación

La planificación original no sufrió cambios estructurales demasiado pronunciados. Todos los problemas surgidos durante el desarrollo no han excedido los límites que se fijaron en un principio, por lo que no ha habido que realizar ningún plan de acción para solventar retrasos, ya que no los ha habido. No obstante, la planificación inicial se ha visto **afectada ligeramente**, no en cuanto a retrasos sino en cuanto a paralelismo. Cuando se realizó la planificación inicial, se establecieron diferentes tipos de tareas, que se pensaron para ser más o menos paralelizables. Sin embargo, en esa planificación esas tareas seguían un orden secuencial con ligeras paralelizaciones.

A medida que avanzaba el proyecto se vio que la implementación de ciertas funcionalidades **podía adelantarse**. Esto se debía a que no ofrecían las dependencias que se había pensado que podrían tener en un principio. El ejemplo más significativo de este aspecto es la red social, a la que se ha dedicado un mayor tiempo del planificado (sin sufrir un retraso en la planificación original). Otro ejemplo de este fenómeno se ha visto en la implementación de la interfaz entre la red y el sistema multi-agente, que se ha podido desarrollar junto a los sistemas que conecta.

En resumen, aún habiendo realizado algunos cambios al orden original de las tareas, la planificación original no sufrió ningún retraso en cuanto a las diferentes etapas. Por tanto, esta planificación inicial fue muy acertada y ha sido de gran ayuda para el desarrollo, aunque después se haya paralelizado más el trabajo.

12.2 Presupuesto

12.2.1 Presupuesto original

Antes de empezar el desarrollo técnico, se estimaron todos los costes que podrían producirse en el proyecto, y se estimaron los gastos mínimos y máximos dependiendo de los posibles retrasos en la finalización de las tareas. Por la naturaleza del proyecto, el presupuesto se llevó a cabo con algunas consideraciones iniciales, que fueron:

- Los costes de los recursos se estimaron para intentar ser lo más realistas posible. Por otro lado, aquellos que podían ser totalmente precisados disponían de precios reales.
- Para que el presupuesto se pareciera lo máximo posible al de un proyecto real, se tuvieron en cuenta costes que realmente no habría que realizar. La mayoría de recursos ya estaban disponibles desde un inicio o sólo serían necesarios si el proyecto fuera comercial.
- Algunos costes se encontraban originalmente en dólares estadounidenses (USD). La conversión de estos se realizó con el ratio de conversión 1 USD = 0.77 EUR a día 3 de julio de 2013.

El siguiente paso era identificar y estimar cada uno de los costes. Debido a que es un proyecto de *software*, el tiempo era el principal factor a tener en cuenta. Los gastos, entonces, se separaron en costes fijos (los que no dependen de retrasos o adelantos) y en costes variables.

Costes fijos

Los costes fijos eran aquellos que sólo hay que realizar una vez. Este tipo de gastos se realizan al empezar el proyecto o al iniciar alguna de las fases de este. Se fijaron diferentes costes de este tipo:

1. **Servidor:** usado para las fases de implementación y pruebas. Se escogió la gama DELL PowerEdgeTM Entry level que podría tener un coste desde 599€ hasta 1799€ y que dependería de la especificación de los sistemas. En el diagrama de Gantt se estimaba el tiempo de implementación y pruebas en 81, por lo que el coste imputable era de entre 44€ y 133€.
2. **Ordenador personal:** usado para desarrollar el *software* y los documentos del proyecto. El coste de adquisición de este ordenador se fijó en 850€. Se

establecieron 3 años de amortización y un uso de 127 días (duración total del proyecto estimado en el diagrama de la figura 12.1). El coste imputable final era de 98€.

3. **Editor de código:** el desarrollo se iba a realizar usando el editor de código Sublime Text 3, el cual dispone de una licencia con un coste de adquisición de 54€ (\$70). Hacienda establece el periodo de amortización de productos de *software* en dos años aproximadamente. El editor se iba a usar durante todo el desarrollo del proyecto, que se estimó anteriormente en 127 días. Por tanto, el coste imputable al proyecto era de 8€.
4. **Framework de visualización de datos:** algunos de los datos recibidos por parte del sistema multi-agente son complejos y necesitas ser representados de forma amigable. Para solucionar esto, se iba a usar el framework Highcharts. Su licencia era gratuita para proyectos no comerciales, pero de 300€ (\$390) si este se volvía comercial. Fijando 2 años de amortización y un uso de 81 días (fase de implementación y pruebas) se obtuvo un coste imputable de 33€.
5. **Otras licencias de software:** se fijó un gasto máximo de 300€ para posibles contingencias.

Costes variables

Los costes variables podían cambiar dependiendo de la duración del proyecto. Para que la estimación del presupuesto fuera útil, había que predecir de la forma más precisa posible el tiempo y coste total de cada uno. Se estimaron los siguientes costes variables:

1. **Mantenimiento del servidor:** el coste de la conexión a la red del servidor y su gasto en electricidad dependen del tiempo que este se use. Para minimizar costes, se contrataría sólo el tiempo estimado de la implementación y fase de pruebas. La planificación estimaba esta duración en 11 semanas y media (81 días), a la se añadió un margen de error del 20 % debido a posibles retrasos o adelantos. A partir de estos datos, se estimó un coste de mantenimiento de entre 178€ y 266€.
2. **Recursos humanos:** el desarrollo real del proyecto ha sido llevado a cabo por un solo estudiante y su director. Sin embargo, se estimó el coste en recursos humanos como si el proyecto hubiera sido realizado por un equipo completo. Se fijó el tiempo dedicado por el estudiante en 18 créditos ECTS³, aproximadamente 540 horas. El total de estas horas se dividió dependiendo del

³European Credit Transfer and Accumulation System

rol realizado en las diferentes etapas planificadas. Por otro lado, las horas del director se estimaron en 10 semanales durante 18 semanas (180 horas en total). La tabla 12.1 muestra la estimación de estos costes.

Tabla 12.1: Coste de los recursos humanos en el proyecto

Salario	Horas	€/Hora	Total
Director	180	35	6300€
Analista/diseñador	165	30	4950€
Programador	220	20	4400€
Diseñador web	85	20	1700€
<i>Tester</i>	70	15	1050€
TOTAL	720	-	17350€

3. **Pruebas con el caminador robótico:** se estimó el coste de las pruebas con el i-Walker en 5€ por cada hora. El tiempo estimado de uso era de unas 30 horas con un margen de error del 20 %, lo que resultaba en un coste total de entre 120€ y 180€.
4. **Otros gastos generales:** durante el proyecto se han producido diferentes gastos generales no relacionados directamente con el desarrollo. Por ejemplo, se ha usado electricidad para alimentar el ordenador personal y una conexión de red. Se estimó el gasto anual de estos aspectos en 500€, lo que llevaba a un gasto de 174€ durante los 127 días de proyecto.

Resumen del presupuesto original

Una vez se finalizó la estimación de cada uno de los costes, se procedió a sumar todos los costes del proyecto. La tabla 12.2 muestra todos ellos, incluyendo la estimación mínima y máxima. Por otro lado, la tabla 12.3 muestra los mismos costes, pero incluyendo la amortización estimada, es decir, el coste real imputable al proyecto. En las tablas se aprecia como el presupuesto original estimó un coste total variable entre **23383€** y **25740€** y un coste real imputable de entre **21746€** y **22435€**.

12.2.2 Cambios en el presupuesto

La estimación de los costes llevada a cabo durante el hito inicial se calculó ligeramente al alza, teniendo en cuenta gastos que podrían no ser necesarios para el tipo de proyecto que se estaba realizando. Esto se ha confirmado a medida que se desarrollaba

Tabla 12.2: Estimación inicial de los costes totales del proyecto (incluyendo adquisiciones)

Tipo	Coste mínimo(€)	Coste máximo (€)
Servidor	599	1799
Ordenador personal	850	850
Editor de código	54	54
Framework de visualización de datos	0	300
Otras licencias de <i>software</i>	0	300
Mantenimiento del servidor	178	266
Recursos humanos	17350	17350
Pruebas con el caminador robótico	120	180
Otros gastos generales	174	174
TOTAL	19325	21273
Impuesto sobre el valor añadido (21 %)	4058	4467
TOTAL (IVA INCLUÍDO)	23383	25740

Tabla 12.3: Estimación inicial de los costes totales imputables al proyecto

Tipo	Coste mínimo(€)	Coste máximo (€)
Servidor	44	133
Ordenador personal	98	98
Editor de código	8	8
Framework de visualización de datos	0	33
Otras licencias de <i>software</i>	0	300
Mantenimiento del servidor	178	266
Recursos humanos	17350	17350
Pruebas con el caminador robótico	120	180
Otros gastos generales	174	174
TOTAL	17972	18542
Impuesto sobre el valor añadido (21 %)	3774	3893
TOTAL (IVA INCLUÍDO)	21746	22435

el proyecto, por lo que los gastos han sido menores de lo previsto. A continuación se enumeran todos los cambios que se han llevado a cabo:

1. **Servidor:** no ha sido necesaria la adquisición de un servidor para desarrollar el proyecto. Por tanto, se elimina el coste de adquisición del servidor, así como el coste de su mantenimiento. Sin embargo, este punto en el presupuesto queda vigente para un futuro, cuando se diera un uso a los sistemas que se han

desarrollado.

2. **Framework de visualización de datos:** se decidió cambiar el framework de visualización por problemas de licencias. Se ha buscado que el proyecto pueda mantener una licencia libre, algo que no podía conseguir con esta herramienta. El nuevo framework (D3.js) sí lo permite, ya que es libre y además carece de coste alguno.
3. **Otras licencias de software:** se estableció un gasto máximo de 300€ para posibles licencias de *software* que pudieran surgir. No obstante, esta parte del presupuesto no ha sido necesaria, ya que al final sólo se han usado bibliotecas gratuitas y de código abierto.

El resto de costes y su estimación se mantienen de la misma forma, ya que los obstáculos encontrados durante el desarrollo no han implicado ningún aumento o disminución de costes. Además, hay que tener en cuenta que el servidor era el elemento del presupuesto que ofrecía una mayor desviación, por lo que ahora la diferencia entre la estimación mínima y máxima será más baja. En las tablas 12.4 y 12.5 se pueden observar las nuevas estimaciones de los costes del proyecto, total e imputable, sucesivamente.

Tabla 12.4: Costes finales totales del proyecto (incluyendo adquisiciones)

Tipo	Coste mínimo(€)	Coste máximo (€)
Ordenador personal	850	850
Editor de código	54	54
Recursos humanos	17350	17350
Pruebas con el caminador	120	180
Otros gastos generales	174	174
TOTAL	18548	18608
Impuesto sobre el valor añadido (21 %)	3721	4467
TOTAL (IVA INCLUÍDO)	22443	23075

12.3 Metodología de desarrollo

En la planificación inicial se fijaron diversas metodologías de trabajo, en las cuales se fijaron criterios para la toma de decisiones y métodos para mantener un buen seguimiento y organización. En las siguientes subsecciones se explicarán estas metodologías iniciales y los cambios que han sufrido durante el proyecto.

Tabla 12.5: Costes finales totales imputables al proyecto

Tipo	Coste mínimo(€)	Coste máximo (€)
Ordenador personal	98	98
Editor de código	8	8
Recursos humanos	17350	17350
Pruebas con el caminador robótico	120	180
Otros gastos generales	174	174
TOTAL	17751	17810
Impuesto sobre el valor añadido (21 %)	3727	3740
TOTAL (IVA INCLUÍDO)	21478	21550

12.3.1 Criterios de validación

En la planificación inicial se fijaron tres validaciones diferentes para los diferentes objetivos del proyecto. Estas se realizarían después de las especificación, diseño e implementación. Durante cada validación se especificaría si se cumplen los requisitos o se enumerarían los diferentes cambios que debían hacerse. En este último caso, se establecería una nueva fecha aproximada para repetir de nuevo la validación. Cuando se consiguiera una validación exitosa, se procedería a iniciar la siguiente etapa del proyecto. Cada validación consistiría en una o más reuniones con el director del proyecto. En estas reuniones se analizaría todo el trabajo realizado y se compararían con los requisitos fijados con anterioridad, lo que decidiría si la validación es exitosa.

12.3.2 Métodos de trabajo y organización

La organización del proyecto original y los métodos de trabajo se basaron en seis elementos principales:

- Reuniones de seguimiento periódicas preestablecidas con el director. En estas reuniones se comentarían los avances realizados y se visualizaría el progreso realizado en comparación con los objetivos. Este tipo de reuniones pueden finalizar con la realización de posibles cambios en las tareas a realizar o el tiempo para realizarlas.
- Reuniones esporádicas para solventar problemas eventuales.
- Comunicación básica a través de correo electrónico.
- La gestión del tiempo y las tareas se realizaría usando la página de planificación

de proyectos Gantter.

- La gestión de la implementación se llevaría a cabo mediante el sistema de versiones Subversion y el sistema de gestión de errores Trac. De esta manera, el director del proyecto podría probar las últimas versiones y comentar errores o mejoras a realizar a medida que avanza el desarrollo.
- Para compartir otros materiales se realizaría una reunión esporádica o se utilizaría un sistema de almacenamiento privado online.

12.3.3 Cambios en las metodologías

Durante el proyecto no ha habido ningún problema relacionado con las metodologías usadas. Por esta razón, no ha sido necesario hacer cambios estructurales en la metodología propuesta en un inicio. No obstante, ha habido algunos cambios en el *software* utilizado. En concreto, en vez de usar Subversion y Trac para el seguimiento de la implementación, se ha hecho uso de un repositorio Git y Github, debido a que se decidió que era una mejor opción y ofrecía más posibilidades. Además de esto, por falta de tiempo, el correo electrónico ha tomado más importancia de la estimada para realizar el seguimiento del proyecto. Sin embargo, como ya se ha comentado, no ha habido que hacer cambios a la estructura general de la metodología original.

12.4 Impacto

El desarrollo de este proyecto puede causar un impacto en diferentes sectores de la sociedad. Este impacto puede ser tanto positivo como negativo. Antes de iniciar el desarrollo se estableció el impacto que podría tener en tres aspectos diferentes: social, ambiental y económico. El objetivo era intentar identificar algún impacto negativo y buscar una manera de reducirlo o eliminarlo. En las siguientes subsecciones se detallan las diferentes estimaciones originales del impacto y cómo reducirlo en caso de que fuera negativo. Además, se han incluido algunos cambios al impacto estimado inicialmente una vez finalizado el proyecto.

12.4.1 Impacto social

Debido a que se va a desarrollar una red social, el impacto puede ser grande para todos los actores relacionados. Entre ellos, el impacto más importante se produciría

en el grupo de pacientes. Esto se debe a que la mayoría serán personas mayores y, por tanto, es muy probable que no estén familiarizadas con los sistemas informáticos. Por tanto, el uso de la red podría provocarles un mayor interés en el mundo de la informática y la electrónica. Adicionalmente, la mejora en su tratamiento médico y seguimiento hará que aumente su calidad de vida. Por otro lado, la conexión entre los pacientes y sus familiares favorecerá la comunicación entre ellos. La mejora en este aspecto puede ser grande, ya que en muchas ocasiones las personas no comunican su malestar con los demás.

Además de sus ventajas, el sistema también podría llegar a presentar inconvenientes. Por ejemplo, es posible que los cuidadores utilicen la red cada vez más como una herramienta única y no como un complemento al seguimiento, reduciendo las relaciones personales entre paciente y cuidador. No obstante, se considera que las posibles ventajas sociales superan en gran medida a las desventajas.

12.4.2 Impacto ambiental

Tratándose de un proyecto que se basa principalmente en el desarrollo de *software*, el impacto ambiental es muy bajo. Todos los aspectos a tener en cuenta para calcular este impacto son:

1. Recursos utilizados durante el desarrollo: ordenadores personales y electricidad.
2. Materiales usados para la construcción de los servidores y recursos energéticos para su mantenimiento. Los materiales usados en el servidor era el mayor impacto ambiental del proyecto. Para intentar reducir este impacto, se escogió una gama de servidores DELL, marca que se encuentra actualmente en la quinta posición de la lista *Guide to Greener Electronics* [27] de Greenpeace. De esta manera intentaba reducir este impacto. No obstante, como se ha visto anteriormente, finalmente no se usó un servidor. Por esa razón, el impacto ambiental ha sido menor del esperado.
3. Materiales y recursos energéticos usados por los servicios asistenciales (caminador i-Walker).

En resumen, el mayor impacto ambiental del proyecto era el uso de un servidor. No obstante, sí sería necesario si el proyecto se usara en producción. En ese caso podrían usarse los servidores DELL que se especificaron en un principio, ya que el impacto ambiental es menor que en otras marcas. Por estas razones, el impacto ambiental del proyecto en su estado actual puede considerarse despreciable.

12.4.3 Impacto económico

El proyecto se llevaba a cabo en un ámbito público y libre, por lo que se no se esperaba ningún beneficio directo. No obstante, en un futuro, la continuación del proyecto podría producir un impacto en forma de ahorro en determinados aspectos médicos. Por ejemplo, se podría reducir el gasto en pruebas médicas y seguimiento del paciente. Además, el estudio de la incorporación de sistemas multi-agente dentro de redes sociales podría ser utilizado para futuros proyectos en otros campos, que tendrán su impacto económico propio.

12.4.4 Viabilidad y sostenibilidad

Debido a que el proyecto no era comercial, no tenía mucho sentido hablar sobre viabilidad, ya que no se esperaba ningún provecho económico. No obstante, los beneficios humanos esperados eran suficientemente altos para justificar los costes. Por tanto, en ese aspecto el proyecto podría considerarse viable.

La sostenibilidad del proyecto en términos económicos, al igual que la viabilidad, tampoco tenía mucho sentido. Como se ha dicho anteriormente, el proyecto no está pensado para ser comercial. Por tanto, después de su finalización sólo se generarían gastos de mantenimiento sin ningún beneficio económico. Aún así, se consideraba una vez más que los beneficios humanos son los suficientes para considerar el proyecto sostenible.

Después de acabar el desarrollo, el proyecto podría ser llevado a producción. En ese caso, los costes deberán controlarse con una rendición de cuentas cada cierto tiempo. El mecanismo utilizado para realizarla se basará en enumerar todos los gastos producidos en renovación de *hardware*, gastos en mantenimiento, recursos humanos y otros gastos. Cada coste deberá ir acompañado de una justificación, para así poder minimizarlo en un futuro o para estudiar si podría aumentar con el tiempo, haciendo el proyecto insostenible.

12.5 Regulaciones

Un punto importante a tener en cuenta en el proyecto son las regulaciones que pueden afectarle. En esta sección se describen todas las regulaciones que pueden afectar el proyecto y si ha habido que quebrantar alguna de ellas.

Debido a que el proyecto consiste en el desarrollo de sistemas informáticos, y que no es posible implementarlo todo desde cero, la mayoría de normativas y regulaciones tienen que ver con el uso de propiedad intelectual de terceros.

Otro tipo de regulaciones que pueden influir incluyen aquellas relacionadas con la privacidad de los datos personales en un sistema médico. Este último punto, sin embargo, no ha sido tenido en cuenta con gran dedicación, ya que las normativas serán diferentes dependiendo del país donde se usen los sistemas. Por tanto, se deja la puerta abierta a los desarrolladores para que cumplan la normativa vigente allí donde se haga uso de los sistemas. A pesar de esto, la sección 12.5.4 cita algunas de las regulaciones de protección de datos que hay que llevar a cabo en este tipo de sistemas en la Unión Europea y en España, normativas que se debería aplicar si el sistema se llevara a producción.

12.5.1 Bibliotecas y frameworks

Debido a la complejidad de algunas partes del proyecto, este hace uso de diferentes bibliotecas, frameworks y modelos de *software*. A continuación se enumeran todos ellos y se especifica su licencia:

- **Elgg**[14]. Es el motor utilizado para desarrollar la red social del proyecto. Elgg dispone de una licencia[19] doble, que incluye las licencias GNU GPL2[25] y MIT[33]. Estas licencias dan la libertad de usar, copiar y modificar el software, por lo que no existen problemas de licencias en ningún caso.

Además del núcleo de Elgg, también se ha hecho uso de algunas modificaciones de terceros. En concreto, se ha extendido una modificación que permite añadir roles y se ha modificado un tema para personalizarlo a las necesidades del proyecto. En ambos casos, la licencia es también GPL2.

Siguiendo con esta filosofía, todas las nuevas modificaciones incluidas en Elgg para realizar este proyecto mantienen la licencia GPL2, por lo que su uso, copia y modificación son libres para cualquiera que quiera utilizarlas.

- Smart Python multi-Agent Development Environment (**SPADE**)[36]. Plataforma de agentes escrita en Python, usada para desarrollar el sistema de Inteligencia Artificial. SPADE está licenciado bajo GNU LGPL 2.1[26]. Esto daría la posibilidad de usar SPADE para desarrollar aplicaciones no licenciadas bajo GPL, pero no es el caso de este proyecto.
- Otras bibliotecas de Python. Aunque la versión de Python requerida (mayor que 2.2) es compatible con GPL, al usar Python el proyecto se expone a hacer

uso de bibliotecas bajo diferentes licencias. En [40] se enumeran algunas de ellas. No obstante, ninguna ofrece ninguna restricción al proyecto.

El proyecto también hace uso de dos módulos de Python que no forman parte del núcleo. En primer lugar se encuentra **Bottle**, un servidor HTPP. **Bottle** está licenciado[6] bajo MIT, que como ya se ha comentado antes, no implica ningún problema. En segundo lugar se encuentra el módulo **requests**, que facilita enviar peticiones a un servidor. Este módulo tiene una licencia Apache 2[4]. La licencia Apache 2 es de código libre, por lo que la biblioteca puede ser usada para el proyecto sin ningún tipo de problema.

- **D3.js**[17] y **NVD3.js**[35]. Usadas para generar gráficos en los informes de la red social. No aparecieron en la planificación inicial, pero se decidió cambiar el framework anterior por sus restricciones en la licencia y por ofrecer menos posibilidades. D3.js está licenciado bajo BSD[9] y NVD3 bajo Apache, licencias libres que permiten usarlos y modificarlos sin problemas.

12.5.2 Algoritmos y paradigmas

El proyecto no hace uso de ningún algoritmo o paradigma patentado. La mayoría de los algoritmos usados son completamente originales. Aquellos que no lo son completamente son algoritmos ampliamente conocidos, bajo los cuales no existe ninguna patente o derechos de autor. Los paradigmas usados no tienen ningún tipo de propiedad intelectual. También hay paradigmas propios (**glsmulti-agent** en una red social), creados a partir de la composición de ideas encontradas en la literatura y otros paradigmas anteriores. En resumen, no existe ningún tipo de problema a nivel algorítmico en el proyecto.

12.5.3 Uso de software

Para finalizar, además de usar código de terceros que formará parte del producto final, también se ha hecho uso de otros tipos de *software* para poder conseguir ese resultado.

1. **Git**. Es el control de versiones utilizado para controlar el código durante el desarrollo del proyecto. Tiene una licencia GNU GPL 2, por lo que se puede hacer un uso libre de él.
2. **APACHE**. Es el servidor usado para albergar la red social, posee la licencia Apache 2[4], que lleva su nombre. Como ya se comentó en la sección anterior,

esta licencia no ofrece al proyecto ninguna limitación.

3. **MySQL.** Es la base de datos usada para albergar los datos en el proyecto. MySQL puede disponer de diferentes licencias. En el caso de este proyecto, se puede hacer uso de la licencia GNU GPL, de la que ya se ha hablado anteriormente.

12.5.4 Protección de datos personales

Existen diversas normativas a tener en cuenta en cuanto a protección de datos personales en un sistema informático que almacene datos médicos. En esta sección se describirán aquellas que debería tener en cuenta un desarrollador para hacer un uso real de los sistemas diseñados. Las normativas estarán divididas entre las de la Unión Europea y aquellas específicas de España.

Unión Europea

De acuerdo al artículo 16 del Tratado de Funcionamiento de la Unión Europea [12], todas las personas tienen el derecho de la protección de sus datos personales. Más en detalle, el artículo 8 de la Carta de los Derechos Fundamentales de la Unión Europea [10] añade que:

1. Los datos personales deben ser tratados de modo leal, para fines concretos y sobre la base del consentimiento de la persona afectada o en virtud de otro fundamento legítimo previsto por la ley. Toda persona tiene derecho a acceder a los datos recogidos que la conciernan y el derecho a que sean rectificados.
2. Respetar estas normas quedará sujeto al control de una autoridad independiente.

El 25 de enero de 2012, la Comisión Europea propuso una revisión [43] de las directivas de protección de datos. Algunos aspectos importantes de la revisión son:

- Fortalecer el derecho a ser olvidado para ayudar a las personas a gestionar mejor sus datos en Internet. Un individuo puede pedir que sus datos personales sean eliminados si así lo desea.
- Garantizar un acceso fácil a los datos personales.
- Establecer un derecho a los individuos para la transferencia libre de datos personales de un servicio a otro (portabilidad de datos).

Esta revisión incluye también cómo se debe guardar y procesar la información de carácter médico para asegurar la privacidad de los pacientes y a su vez poder usarla

para mantener un alto nivel de protección sanitaria (investigación, diagnóstico, etc.). Las normativas más importantes en este aspecto son:

- Los datos personales de carácter médico deben incluir datos concretos pertenecientes al estado de salud del sujeto, incluyendo: información de la inscripción del individuo para la prestación de servicios médicos, un símbolo para identificar al individuo de forma única, cualquier información de la persona durante las pruebas realizadas, etc.
- El procesado de información personal sanitaria puede ser justificado por diferentes razones legítimas, incluyendo el beneficio de individuos o de la sociedad en su conjunto, especialmente en el contexto de asegurar la calidad de la asistencia sanitaria.
- El procesado de información personal puede ser llevado a cabo sin el consentimiento explícito de la persona implicada por razones de interés público en áreas de salud pública.

España

En España, la protección de datos personales viene gestionada por la Ley Orgánica de Protección de Datos de Carácter Personal (LOPD). Esta ley tiene el objetivo de proteger el tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, su honor, intimidad y privacidad personal y familiar.

La última actualización de esta ley se llevó a cabo con el Real Decreto 1720/2007, de 21 de Diciembre de desarrollo de la Ley Orgánica de Protección de Datos [42]. La nueva ley actualiza la anterior de 1999, añadiendo gran cantidad de artículos (pasando desde los 29 a los 158 artículos). De entre ellos, los más importantes se encuentran en el título VIII (artículos 79 a 114). Estos artículos definen las medidas de seguridad que hay que llevar a cabo separadas en tres niveles (bajo, medio y alto), que deberán aplicarse dependiendo del tipo de datos personales usados.

Los sistemas que traten con datos médicos, como es el caso de este proyecto, deberán ofrecer unas medidas de seguridad de nivel alto. Además de cumplir esas medidas, también se deben aplicar las de los niveles inferiores. Es decir, un sistema que requiera seguridad de nivel alto deberá también ofrecer las de nivel medio y bajo. Las medidas separadas por niveles son:

1. **Nivel bajo:** se debe llevar un registro de las incidencias que afecten a información personal (artículo 90), realizar un control de acceso a los datos personales (artículo 91) y un control de los soportes y documentos (artículo 92). Además,

los usuarios deben necesitar de autenticación para acceder a los datos (artículo 93). Finalmente, deben realizarse copias de respaldo al menos de forma semanal (artículo 94).

2. **Nivel medio:** se debe asignar un responsable de seguridad (artículo 95) y realizarse una auditoría de seguridad cada dos años (artículo 96). Además, debe llevarse a cabo un control de acceso físico a los equipos que dispongan de la información (artículo 99).
3. **Nivel alto:** Debe existir un registro de acceso detallado (artículo 103). El registro debe contener como mínimo el usuario, la fecha y la hora del acceso. Además, el registro debe ser revisado al menos una vez al mes. Por otro lado, las comunicaciones donde se transmita información personal deben estar cifradas para que no puedan ser leídas o modificadas por terceros.

A

Instalación

En este anexo se describe cómo instalar y ejecutar los diferentes sistemas en la distribución Debian de Linux. La instalación en otros sistemas operativos puede diferir en algunos aspectos (sobre todo en lo referente a rutas y comandos), pero los pasos a seguir serán los mismos para todos ellos.

La implementación está separada en dos directorios: **Elgg** y **SPADE**, que contienen la red social y el sistema multi-agente, respectivamente. Cada una de estas partes requiere de procesos diferentes para su instalación, que se describen en las secciones siguientes.

A.1 Red social

Para poder instalar el motor de la red social (Elgg), primero es necesario que en la máquina esté corriendo un servidor Apache con PHP (versión mayor o igual a 5.2) y MySQL (versión igual o mayor que 5). Además, es necesario crear una base de datos antes. Aquí se va a suponer que la red social dispondrá de un dominio local **rssapm.dev** y una base de datos de nombre **rssapm**. Entonces, dentro del directorio del dominio hay que copiar integralmente la carpeta de nombre **Elgg**.

El siguiente paso es instalar la base del motor de la red. Es tan fácil como ejecutar (con un navegador Web) el fichero **install.php** dentro que la carpeta que se copió al dominio. Siguiendo la nomenclatura, se encontrará en **rssapm.dev/Elgg/install.php**. Esta instalación realizará una serie de pasos:

1. Mensaje de bienvenida
2. Comprobación de requisitos: comprobará que la versión de PHP y la configuración de Apache sea la correcta.
3. Configuración de la base de datos
4. Configuración del sitio: permite fijar algunas configuraciones para el sitio, tales como nombre, dirección, etc. Hay que fijar un directorio para guardar los datos de los usuarios (fotografías de perfil, etc.), y además, fijar los permisos de nuevo contenido a **private**.

5. Cuenta de administrador: datos para la cuenta de superadministrador, que se encargará de crear los primeros usuarios y de gestionar las modificaciones creadas para el proyecto.
6. Final: ya ha acabado la instalación del motor, sólo hace falta ir al sitio haciendo clic en el botón **Go to site**.

En este momento, nos encontraremos identificados con el usuario superadministrador en el panel de administración. El siguiente paso a seguir es activar las características desarrolladas para la red. Para ello, hay que acceder a la pestaña **plugins**. En esta ventana es necesario ir activando, una a una, todas las modificaciones necesarias hasta lograr activar *Red social como interfaz de servicios asistenciales para personas mayores*, que depende de todas las modificaciones desarrolladas. Algunas de ellas disponen también de dependencias, por lo que será necesario moverlas de sitio antes de poder activarlas.

Una vez completado el anterior paso, la red social ya estará preparada, pero se encuentra en inglés. Para cambiar el idioma (ya que no se incluyen traducciones al inglés de las modificaciones), hay que ir a la pestaña *Settings*, *Basic Settings* y cambiar el idioma predeterminado al español. Esto cambiará el idioma de los nuevos usuarios, pero no del actual. Para ello, hay que ir al sitio (utilizando el panel superior) y hacer clic de nuevo en *Settings*. La opción del idioma aparece al final de esta página.

Por último, sólo falta ir al gestor de la red (pestaña en el menú superior) y empezar a crear nuevos usuarios y relacionarlos entre ellos. También se deben crear nuevos dispositivos asistenciales y, para finalizar, asignarlos a alguno de los pacientes.

A.2 Sistema multi-agente

Antes de poder iniciar el sistema multi-agente, se tiene que comprobar que la versión de Python sea igual o superior a la 2.7.5. Además, también es necesario tener instalado el gestor de paquetes **pip** [38]. En caso de que la versión de Python no exista o sea inferior, debe ser instalada la versión correcta. Sucede lo mismo con **pip**, aunque este último no tiene una versión mínima.

```
python -V
pip -V
```

Antes de poder iniciar los sistemas, es necesario instalar algunos paquetes que se usan en la implementación. Estos paquetes son:

1. **SPADE**: plataforma para el sistema multi-agente.

2. **Bottle**: servidor HTTP usado para implementar la interfaz en el lado del sistema multi-agente.
3. **Requests**: módulo que facilita el envío de peticiones HTTP para comunicarse con la red social.

Estos módulos pueden instalarse con los siguientes comandos:

```
pip install SPADE  
pip install bottle  
pip install requests
```

En este momento, todos los componentes ya están instalados. El siguiente paso es configurar algunos parámetros para que el sistema multi-agente pueda funcionar correctamente. Para ello, hay que abrir el fichero `config.py` que se encuentra dentro del directorio `rssapm` en el sistema multi-agente. Este fichero de configuración contiene puertos y direcciones necesarios para comunicarse con la red social. En caso de que ambos sistemas se encuentren en la misma máquina, sólo habría que cambiar la variable `elgg_server`, haciendo que apunte a la dirección base de la red social, que puede ser diferente dependiendo de la instalación de Apache.

Ahora solo falta ejecutar los componentes necesarios. Para lograrlo, se necesitarán tres consolas `tty` distintas. La primera de ellas ejecutará la plataforma de agentes. La segunda se encargará de los agentes que se encuentran en el servidor. Por último, la tercera ejecutará el agente del dispositivo de ejemplo (i-Walker). En este anexo, el dispositivo está funcionando en el mismo servidor, usando archivos ya generados anteriormente. En la realidad, la última consola debería ejecutarse en el propio dispositivo, teniendo este que instalar y configurar el sistema multi-agente de la misma forma. La ejecución de cada parte es sencilla. Existen tres *scripts* ya creados para realizar todas las acciones, así que sólo hay que moverse al directorio `SPADE` y ejecutar cada uno de los *scripts* en las diferentes consolas por orden. Primero `start_spade.sh` para iniciar la plataforma de agentes, después `start.py` para iniciar el gestor y los agentes en el servidor y finalmente `start_devices.py` para iniciar los dispositivos. Hay que tener en cuenta que, en este momento, la red social debe estar funcionando correctamente, ya que es necesaria para iniciar los agentes. Los últimos comandos necesarios para iniciar los sistema son, por tanto:

```
cd SPADE  
sh start_spade.sh  
python start.py  
python start_devices.py
```

Bibliografía

- [1] Roberta Annicchiarico y col. «The i-walker: an intelligent pedestrian mobility aid». En: *IOS Press* 178 (2008), págs. 708-712.
- [2] *Apache HTTP Server Project*. URL: <http://httpd.apache.org/> (visitado 25-11-2013).
- [3] *Apache HTTP Server Tutorial: .htaccess files*. URL: <http://httpd.apache.org/docs/2.2/howto/htaccess.html> (visitado 25-11-2013).
- [4] *Apache License Version 2.0*. URL: <http://www.apache.org/licenses/LICENSE-2.0.html> (visitado 25-11-2013).
- [5] Fabio Bellifemine, Agostino Poggi y Giovanni Rimassa. «Developing multi-agent systems with a FIPA-compliant agent framework». En: *Software-Practice and Experience* 31.2 (2001), págs. 103-128.
- [6] *Bottle License*. URL: <http://bottlepy.org/docs/dev/#license> (visitado 25-11-2013).
- [7] Danah m. Boyd y Nicole B. Ellison. «Social Network Sites: Definition, History, and Scholarship». En: *Journal of Computer-Mediated Communication* 13.1-2 (2007).
- [8] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Cambridge University Press, 1999. ISBN: 1575861925.
- [9] *BSD License*. URL: <http://opensource.org/licenses/BSD-3-Clause> (visitado 25-11-2013).
- [10] *Charter of fundamental rights of the European Union*. URL: <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:C:2000:364:0001:0022:EN:PDF> (visitado 14-01-2014).
- [11] *Community Engine*. URL: <http://communityengine.org/> (visitado 02-12-2013).
- [12] *Consolidated version of the treaty on the functioning of the European Union*. URL: <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:C:2008:115:0047:0199:EN:PDF> (visitado 14-01-2014).
- [13] Ulises Cortés y col. «Assistive technologies for the new generation of senior citizens: the share-it approach». En: *IJCIIH* (2010), págs. 35-65.
- [14] Cash Costello. *Elgg*. URL: <http://elgg.org/about.php> (visitado 25-11-2013).
- [15] Cash Costello. *Elgg Datamodel*. URL: <http://docs.elgg.org/wiki/Engine/DataModel> (visitado 25-11-2013).
- [16] Cash Costello. *Elgg Plugin Development*. URL: http://docs.elgg.org/wiki/Plugin_development (visitado 25-11-2013).
- [17] *Data-Driven Documents*. URL: <http://d3js.org/> (visitado 25-11-2013).
- [18] *Drupal: Open Source CMS*. URL: <https://drupal.org/> (visitado 02-12-2013).

- [19] *Elgg License*. URL: <http://elgg.org/license.php> (visitado 25-11-2013).
- [20] *FIPA ACL Message Structure Specification*. URL: <http://www.fipa.org/specs/fipa00061/SC00061G.html> (visitado 25-11-2013).
- [21] *FIPA Agent Communication Language*. URL: <http://www.fipa.org> (visitado 25-11-2013).
- [22] *FIPA Agent Management Specification*. URL: <http://www.fipa.org/specs/fipa00023/XC00023H.html> (visitado 25-11-2013).
- [23] *FIPA Communicative Act Library Specification*. URL: <http://www.fipa.org/specs/fipa00037/SC00037J.html> (visitado 25-11-2013).
- [24] Ricard Gavaldà, Marta Arias y José L. Balcázar. «Slides of CAIM subject, UPC. Network Analysis».
- [25] *GNU General Public License 2*. URL: <http://www.gnu.org/licenses/gpl-2.0.html> (visitado 25-11-2013).
- [26] *GNU Lesser General Public License 2*. URL: <http://www.gnu.org/licenses/lgpl-2.1.html> (visitado 25-11-2013).
- [27] Greenpeace. *Guide to Greener Electronics*. 2013. URL: <http://www.greenpeace.org/international/en/campaigns/climate-change/cool-it/Campaign-analysis/Guide-to-Greener-Electronics/> (visitado 25-11-2013).
- [28] Fumio Hattori y col. «Socialware: Multiagent Systems for Supporting Network Communities». En: *Communications of the ACM* 42 (1999), págs. 55-61.
- [29] *I-DONT-FALL project*. URL: <http://www.idontfall.eu/> (visitado 25-11-2013).
- [30] *i-Walker: concept*. URL: <http://www.diagnostic-walker.es/page4/page4.html> (visitado 14-01-2014).
- [31] *Jade (Java Agent Development Framework)*. 2013. URL: <http://jade.tilab.com/> (visitado 11-07-2013).
- [32] *Magentix 2*. URL: <http://www.gti-ia.upv.es/sma/tools/magentix2/> (visitado 11-07-2013).
- [33] *MIT License*. URL: <http://opensource.org/licenses/mit-license.php> (visitado 25-11-2013).
- [34] *MySQL, The world's most popular open source database*. URL: <http://www.mysql.com/> (visitado 25-11-2013).
- [35] *NVD3*. URL: <http://nvd3.org/> (visitado 25-11-2013).
- [36] Javier Palanca. *SPADE (Smart Python multi-Agent Development Environment)*. URL: <https://github.com/javipalanca/spade> (visitado 11-07-2013).
- [37] *PHP: Hypertext Preprocessor*. URL: <http://php.net/> (visitado 25-11-2013).
- [38] *PIP: A tool for installing and managing Python packages*. URL: <https://pypi.python.org/pypi/pip> (visitado 08-12-2013).
- [39] Martha E. Pollack. «Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment». En: *AI magazine* 26.2 (2005), pág. 9.

- [40] *Python Licenses*. URL: <http://docs.python.org/2/license.html#licenses-and-acknowledgements-for-incorporated-software> (visitado 25-11-2013).
- [41] Anand S. Rao y Michael P. Georgeff. «BDI Agents: From Theory to Practice». En: *In proceedings of the first international conference on multi-agent systems (ICMAS-95)*. 1995, págs. 312-319.
- [42] *Real Decreto 1720/2007, de 21 de Diciembre de desarrollo de la Ley Orgánica de Protección de Datos*. URL: <http://www.boe.es/boe/dias/2008/01/19/pdfs/A04103-04136.pdf> (visitado 14-01-2014).
- [43] *Regulation of the European Parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation)*. URL: http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf (visitado 14-01-2014).
- [44] *RESTful Web services: The basics*. URL: <http://www.ibm.com/developerworks/webservices/library/ws-restful> (visitado 25-11-2013).
- [45] Sokichi Sakuragi y Yoshiki Sugiyama. «Effects of daily walking on subjective symptoms, mood and autonomic nervous function.» En: *J Physiol Anthropol* 25.4 (2006), págs. 281-9. ISSN: 1880-6791.
- [46] Gilson Yukio Sato, Hilton José Azevedo y Jean-Paul A Barthés. «Agent and multi-agent applications to support distributed communities of practice: a short review». En: *Autonomous Agents and Multi-Agent Systems* 25.1 (2012), págs. 87-129.
- [47] *SPADE: An agent with a behavior*. URL: <http://pythonhosted.org/SPADE/spade.basicagents.behav.html> (visitado 27-11-2013).
- [48] *SPADE: Sending and Receiving Messages*. URL: <http://pythonhosted.org/SPADE/spade.basicagents.agentmodel.communication.html> (visitado 27-11-2013).
- [49] *SPADE: The Knowledge Base*. URL: <http://pythonhosted.org/SPADE/spade.bdi.kb.html> (visitado 27-11-2013).
- [50] Michael Wooldridge. *An Introduction to MultiAgent Systems*. 2.^a ed. 2009. ISBN: 0470519460.
- [51] Michael Wooldridge y Nicholas R. Jennings. «Intelligent Agents: Theory and Practice». En: *Knowledge Engineering Review* 10 (1995), págs. 115-152.
- [52] Wordpress.org. *BuddyPress*. URL: <http://codex.buddypress.org/> (visitado 11-07-2013).
- [53] XMPP Standards Foundation. *About XMPP*. URL: <http://xmpp.org/about-xmpp/> (visitado 25-11-2013).

Glosario

API	Del inglés <i>Application Programming Interface</i> . Conjunto de componentes <i>software</i> que especifican cómo se deben comunicar dos sistemas informáticos.
Broadcast	Emisión de un mensaje para un público generalizado. En un sistema informático es un mensaje transmitido a todos los sistemas en una misma red.
Escalabilidad	Habilidad de un sistema, red o proceso de manejar un aumento de trabajo de forma satisfactorio o su habilidad de ser ampliado para poder acomonarlo a la nueva cantidad de trabajo.
Framework	Abstracción de <i>software</i> en la cual se provee una funcionalidad genérica, que puede usada para acelerar el desarrollo de código con fines más específicos.
Grafo	Representación de un conjunto de objetos (V) donde algunos pares de ellos están conectados por enlaces (E).
Ingeniería del conocimiento	Disciplina dentro de la Inteligencia Artificial. Usa diferentes tecnologías de la información para representar el conocimiento y razonamiento de los seres humanos dentro de un dominio.
i-Walker	Caminador inteligente que ayuda a caminar a la persona que lo esté usando. Cuenta con diferentes sensores (velocidad, fuerza, etc.) y actuadores (motores) para llevar a cabo sus objetivos.

Raspberry Pi	Microcomputador del tamaño de una tarjeta de crédito. Su tamaño, su poco consumo y sus conexiones lo hacen ser muy útil para proyectos de electrónica.
Red social	Estructura social formada por diferentes actores y un conjunto de conexiones entre ellos.
Sistema multi-agente	Sistema informático compuesto por múltiples agentes inteligentes interactuando en un entorno. Los sistemas multi-agente pueden ser usados para resolver problemas difíciles para sistemas monolíticos (un único agente).
UNIX <i>timestamp</i>	También llamado POSIX time. Sistema para representar el tiempo usando un número entero. Este indica el número de segundos transcurridos desde la medianoche del 1 de enero de 1970 en tiempo universal coordinado.

Siglas

AID	Agent Identifier.
AMS	Agent Management System.
BDI	Belief-desire-intention.
CMS	Content Management System.
CSS	Cascading Style Sheets.
DF	Directory Facilitator.
DNI	Documento Nacional de Identidad.
ECTS	European Credit Transfer and Accumulation System.
EUR	Euro.
FIPA	Foundation for Intelligent Physical Agents.
FIPA-ACL	FIPA Agent Communication Language.
FIPA-SL	FIPA Semantic Language.
FTP	File Transfer Protocol.
GPL	General Public License.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
IA	Inteligencia Artificial.
IDE	Integrated Development Environment.
IVA	Impuesto sobre el Valor Añadido.
JSON	JavaScript Object Notation.
KEML	Knowledge Engineering and Machine Learning group.
LAMP	Linux, Apache, MySQL, PHP.
LEDS	Light Emitting Diodes.

MAS	Multi-Agent System.
MIT	Massachusetts Institute of Technology.
MVC	Model-view-controller.
PHP	PHP: Hypertext Preprocessor.
RDF	Resource Description Framework.
RESTful	Representational State Transfer.
RSS	Rich Site Summary.
SHA1	Secure Hash Algorithm Version 1.
UPC	Universitat Politècnica de Catalunya.
USD	United States Dollar.
XML	Extensible Markup Language.
XMPP	Extensible Messaging and Presence Protocol.