ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

SISTEMA MULTIAGENTE PARA LA SIMULACIÓN DE EPIDEMIAS

MULTIAGENT SYSTEM FOR EPIDEMIC SIMULATION

Realizado por
Antonio José Reina González
Tutorizado por
Eduardo Guzmán de los Riscos
María Victoria Belmonte Martínez
Departamento
Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre de 2015

Fecha defensa:
El Secretario del Tribunal

Resumen: Este trabajo de fin de grado tiene como objetivo realizar un sistema multiagente para simular el desarrollo de las enfermedades epidemiológicas en un entorno concreto. Para ello se plantea hacer un servidor que haga una simulación, especificándole diversos parámetros del entorno, de la enfermedad y otros propios de la simulación. Estos parámetros se pueden especificar desde una aplicación web y desde una aplicación de escritorio. También se podrá visualizar esta simulación desde las dos aplicaciones, una vez que la simulación haya finalizado. Se decide estructurar el sistema de esta forma para dejar la mayor parte del cómputo en manos del servidor. El software se desarrolla íntegramente en Java, haciendo así que sea multiplataforma.
Para el desarrollo de este proyecto se ha investigado sobre la programación orientada a agentes y sobre los distintos modelos de epidemias existentes.
Este es un proyecto grupal, formado por dos compañeros y yo. Ha sido un arduo trabajo de análisis, diseño, implementación y prueba del software por parte de todos. Para facilitar todo este proceso, la sincronización y el reparto de tareas se sigue una metodología de desarrollo ágil.

Palabras claves: Multiagente, epidemia, simulación, JADE, JSF, enfermedad, agente, Java, aplicación web, aplicación escritorio y servicios RESTful.

Abstract: This final project degree aims to conduct a multi-agent system to simulate the development of epidemiological diseases in a particular environment. The proposition will be to make a server that does a simulation, specifying different environmental parameters, diseases and others about the simulation itself. These parameters could be specified from a web and a desktop application. You could also display this simulation from the two applications, once the simulation is completed. It was decided to structure the system in this way in order to make the most of the computing server hands. The software is developed entirely in Java, thus making it multiplatform.

For the development of this project it has been investigated about oriented agents programming and existing models of epidemics.

This is a group project, created by me and two colleagues which has been a hard work of analysis, design, implementation and testing of software by all of us. To facilitate this process, the timing and task sharing agile development methodology has been strictly followed.

Keywords: Multiagent, epidemic, simulation, JADE, JSF, disease, agent, Java, web aplication, desktop aplication and RESTful services.

Índice

Índice de Tablas

Índice de Ilustraciones

1. Introducción

En esta primera sección, vamos a exponer al lector las nociones fundamentales y conceptos de la tecnología que vamos a usar, y a describir brevemente los aspectos principales de este proyecto. En primer lugar, vamos a explicar cuál es la motivación y los objetivos de este trabajo de fin de grado; en segundo lugar, vamos a comentar algún estudio relacionado; por último, se comentarán la secciones en las que se divide la memoria, junto con una breve descripción de las mismas.

1.1. Motivación y Objetivos

El pasado año, un brote de ébola afectó principalmente a la población africana, aunque aparecieron brotes en distintas partes del planeta. El brote comenzó en diciembre de 2013 en Guinea y se extendió posteriormente a Liberia, Sierra Leona y Senegal. Este fue el mayor brote epidémico de la enfermedad producida por el virus del ébola.

Todos estos hechos hicieron que las personas se preocuparan ante una posible pandemia y pusieron de manifiesto que era necesario disponer de mecanismos que permitieran estudiar el impacto de una enfermedad en una determinada población, para saber cómo actuar de forma adecuada frente a ella y minimizar los efectos en la medida de lo posible.

De esta necesidad surgió la idea de modelar, con sistemas multiagente, una aplicación para poder simular el comportamiento de una enfermedad en una población concreta, simplemente introduciendo distintos parámetros tanto de la enfermedad como del entorno.

El objetivo principal es desarrollar un sistema multiagente para la simulación de enfermedades epidémicas que permita el comportamiento característico de una población concreta que ha sido contagiada. Este sistema podrá servir como medio para estudiar el comportamiento de una determinada enfermedad en las distintas situaciones que puedan darse, además de ayudar en la toma de decisiones de cara a paliar sus efectos en la población o establecer medidas de control y seguridad apropiadas para evitar su expansión.

Para lograr los objetivos mencionados, la herramienta se deberá implementar haciendo que el sistema sea dinámico, para que el usuario pueda ajustar los distintos parámetros del entorno y de la enfermedad, y hacer que la simulación sea lo más parecida con la realidad.

También se podrán modificar otros parámetros respecto a la simulación en sí, como puede ser el número de infectados iniciales o el número de días de simulación.

La idea es hacer que el cómputo del sistema se haga todo en un servidor web para que, tanto la aplicación de escritorio como la aplicación web que se desarrollarán en el marco del proyecto, dejen la mayor parte del trabajo y del procesamiento en manos del servidor.

1.2. Estudios Relacionados

Hay un estudio estrechamente relacionado con nuestro trabajo publicado en JASSS (*The Journal of Artificial Societies and Social Simulation*) y titulado *An Agent-Based Spatially Explicit Epidemiological Model in MASON*, realizado por Jill Bigley Dunham, en el que se explica el comportamiento de una aplicación de simulación de epidemias basada en agentes y desarrollada en MASON.

En este estudio se muestra cómo, introduciendo diversos parámetros (como la distancia de exposición, el modelo de epidemia [SIR o SEIR], el número de personas, el número de infectados, la duración de la simulación, etcétera), se puede visualizar el entorno donde los agentes se mueven libremente y distintas gráficas de la evolución de la enfermedad en dicho entorno. En este estudio se puede visualizar la simulación de tres enfermedades: RSV, la gripe y el ébola.

Es muy interesante este estudio, ya que se muestran los distintos diagramas de actividad en los que se basan los agentes humanos a lo largo de la simulación.

Basándonos en él, nuestra intención ha sido desarrollar un sistema similar, pero más configurable y flexible.

1.3. Organización de la Memoria

Capítulo 2. Agentes, Simulación y Modelos de Epidemia

En este capítulo, se explica brevemente la historia y evolución, tanto de los agentes como de la simulación. También se exponen los distintos modelos de epidemias utilizados para la aplicación.

Capítulo 3. Herramientas y Tecnologías

Aquí se exponen las distintas alternativas de herramientas y tecnologías, y se explican con mayor detalle aquellas que han sido utilizadas durante el desarrollo de la aplicación.

Capítulo 4. Análisis

En este capítulo se realiza un estudio de la metodología de desarrollo que se va a utilizar durante el proyecto, se concretarán los requisitos, tanto del sistema multiagente como de las aplicaciones que se van a realizar, así como la descripción de los casos de uso. También se analizará la base de datos y la interfaz de usuario.

Capítulo 5. Diseño

Durante este capítulo, se explicarán distintos diagramas de las diferentes aplicaciones. Empezaremos con un diagrama de distribución, seguido de los diagramas de clase cada una de las aplicaciones y, para terminar, se explicará el modelo relacional de la base de datos.

Capítulo 6. Tareas Realizadas e Implementación

A lo largo de este capítulo, se mencionará la estructura del proyecto y cuáles son las tareas que he realizado, haciendo énfasis en las tareas que me han resultado más costosas y/o me han parecido más interesantes. Para terminar, se comentarán algunos de los problemas que ha habido durante la implementación.

Capítulo 7. Pruebas

Aquí se expondrán las pruebas más relevantes que han sido realizadas al sistema.

Capítulo 8. Conclusiones y Mejoras Futuras

En este último capítulo se explicará cuáles han sido las conclusiones después de realizar todo el proyecto y los posibles desarrollos futuros que podrían mejorar las simulaciones.

2. Agentes, Simulación y Modelos de Epidemia

2.1. ¿Qué es un Agente? La Evolución de los Agentes

El concepto de *agente software* es un poco confuso, ya que existen múltiples definiciones de agente. Agente proviene del latín *agens*, que significa 'hacer'. En la RAE los significados que más se le pueden acercar son: "persona o cosa que produce un efecto" o "persona que obra con poder de otra". Muchas veces la palabra agente por sí sola no dice mucho, más bien necesita de otra para completar su significado como agente comercial, agente de bolsa… En nuestro caso el Agente Software.

Lo que hemos podido sacar en claro hasta ahora es que un agente tiene una característica esencial, el funcionamiento *autónomo*. Un agente tiene que tener capacidad de decisión por sí solo. Otras características comunes que hay en las distintas definiciones de agente son:

- Funcionamiento *continuo*.
- *Comunicación con el entorno y con otros agentes*, ya sean humanos o software, con algún tipo de lenguaje.
- *Robustez.*
- *Adaptabilidad,* pudiendo realizar objetivos y tareas en distintos dominios de forma incremental y flexible.

En pleno auge de la IA aparecen las primeras polémicas sobre las limitaciones de los sistemas expertos, ya que la resolución de problemas se limita a tareas concretas en terrenos restringidos y estos necesitan adquirir o intercambiar datos con el usuario y otras aplicaciones, pero son incapaces de intercambiar conocimientos con otros sistemas expertos.

Las primeras investigaciones sobre los agentes comenzaron a finales de los años setenta y principio de los ochenta. En los primeros estudios se propone la cooperación entre distintos sistemas de resolución de problemas. La cooperación permite a dos o más sujetos realizar colectivamente tareas que no pueden ser realizadas individualmente o resolver problemas de mayor envergadura con más eficiencia que una sola entidad.

Una de las primeras soluciones propuestas fue la arquitectura de pizarra compartida, la cual consiste en la operación entre agentes (aún no eran agentes tal y como los conocemos) que se comunican mediante la pizarra. Cada "agente" tiene parte del conocimiento necesario para resolver un problema. Cada una de ellas coge los datos de la pizarra, los trabaja y, una vez resuelto el problema parcial, los deja en la pizarra. Uno de los mayores problemas de esto es que no solo basta con

producir la solución de los problemas parciales, sino que es necesario en el orden y en el instante adecuado.

En los 80 aparece una nueva tendencia en la que ahora el comportamiento de los agentes es puramente reactivo: reciben eventos procedentes del entorno y realizan acciones según el evento recibido y el estado interno de este. Ahora la importancia no está en las capacidades individuales de los agentes, sino en las interacciones de las cuales emerge el comportamiento global. Esta nueva idea de agente tuvo buena acogida en áreas de simulación de sistemas biológicos, la sociológica, la gestión de procesos industriales o la gestión empresarial. Hay estudios que proponen inspirarse en los modelos de organizaciones humanas y biológicas y aplicar las teorías de la organización.

En esta década es cuando se empiezan a estudiar los principales elementos del proceso de comunicación como: qué es lo que se quiere decir; emisor; receptor o receptores; protocolo, a los distintos niveles; paradigma; y lo que los receptores entienden.

Es en los 90 cuando se empiezan a desarrollar los primeros asistentes inteligentes o agentes de interfaz. El objetivo de estos agentes es que el usuario puede encomendar a estos agentes parte de sus labores como pueden ser reuniones, citas, contestación automática del correo, etc. Estas aplicaciones, aún siendo prototipos, demuestran el potencial de los agentes.

Es también en esta época cuando empiezan a darse los primeros agentes móviles, que son aquellos que tienen la capacidad de moverse entre los diferentes nodos de una red para realizar distintas tareas. Una de las ventajas de estos agentes es que, si necesitan de un uso intensivo de los recursos remotos, se pueden desplazar a otro nodo con mayor capacidad de cómputo, ahorrándose los procesos de comunicación.

Es a finales del siglo XX cuando se empieza a hablar sobre el paradigma de la programación orientada a agentes. También se empiezan a desarrollar lenguajes como KQML (*Knowledge Query and Manipulation Language*) que es un lenguaje y protocolo para la comunicación entre agentes software y KIF (*Knowledge Interchage Format*). Este lenguaje es utilizado por los americanos, pero en Europa se apoya la definición de un nuevo lenguaje denominado FIPA, que engloba a grupos industriales y de investigación con el objetivo de estandarizar los modelos y tecnologías de agentes.

En la actualidad los agentes están integrados en muchas plataformas y hay muchas herramientas de desarrollo orientada a agentes como pueden ser JADE, MadKit, MASON, etc. Actualmente se utilizan en numerosos dominios de aplicación e investigación, y pueden ser una forma muy interesante y conveniente para la

comprensión, modelado, diseño e implementación de diferentes tipos de sistemas distribuidos. También son muy utilizados para las simulaciones por ordenador que es una forma de diseñar, probar y estudiar tanto teorías como sistemas reales para diversos fines.

2.2. La Evolución de las Técnicas de Simulación

En un estudio publicado en la revista EMPIRIA nº 14, edición julio-diciembre de 2007 denominado *Simulación de procesos sociales basada en agentes software*, en el apartado 2, se menciona la evolución de las distintas técnicas de simulación hasta los sistemas multiagentes, la cuales se van a comentar en este apartado. Se han contrastado las distintas técnicas e intentado aclarar los distintos conceptos.

Los primeros modelos relacionados con las simulaciones actuales surgieron en la primera mitad del siglo XX con la teoría de juegos. Es un área de la matemática aplicada que utiliza modelos para estudiar interacciones en estructuras formalizadas de incentivos y llevar a cabo procesos de decisión. Sus investigadores estudian las estrategias óptimas así como el comportamiento previsto y observado de individuos en juegos. Fue desarrollada como una herramienta para entender el comportamiento de la economía aunque después fue usada en muchos campos como la biología, sociología, etc.

A finales de los 40 aparecen los autómatas celulares, teoría iniciada con John Von Neumann. Es un modelo matemático que simula sistemas dinámicos que evolucionan en pasos discretos. Esta técnica consiste en una rejilla o cuadrícula de enteros, donde cada celda de la cuadrícula se conoce como “célula”. Cada célula posee un conjunto finito de estados. Otra característica es que toda célula tiene un conjunto de células vecinas, esto se denomina “vecindad”. También existe una función de transición que tiene como argumentos la célula que va a transitar y los valores de sus vecinos y esta devuelve el valor que tendrá la célula en la etapa posterior.

En los 60 con el gran avance en los ordenadores comenzó el desarrollo de las simulaciones por ordenador. Fueron desarrollados distintos tipos de programas para simulaciones de propósito general. Keith Douglas Tocher desarrolló uno para simular el funcionamiento de una planta de producción. Este trabajo originó el primer libro publicado sobre simulación: The Art of Simulation. Entre 1961 y 1963 distintas empresas como IBM o RAND CORPORATION crearon también software de simulación de propósito general. También Royal Norweigian Computing Center inició el desarrollo de SIMULA obteniendo como resultado el lenguaje de simulación SIMULA I (1962) que fue el primer “lenguaje orientado a objeto” que introdujo el concepto de clase, aunque el primero que tiene todas las características propias de programación orientada a objeto fue SIMULA 67, lanzado posteriormente en 1967.

Muchas de las primeras simulaciones realizadas en ordenador se basan en la dinámica de sistemas, que consiste en la utilización de grandes sistemas de ecuaciones diferenciales para representar las trayectorias de las variables en el tiempo. El gran problema de este método es que solo es posible modelar aquello que pueda expresarse mediante ecuaciones.

Con la aparición de los ordenadores también se originaron simulaciones derivadas de procesos no deterministas como la teoría de colas o el modelado de simulación multi-nivel. La teoría de colas es el estudio matemático en el que se estudian elementos como el tiempo medio de espera en estas, la capacidad de estas, el tipo de cola (LIFO, FIFO…). La simulación multinivel se basa en modelos estadísticos de parámetros que varía en más de un nivel. Estos modelos normalmente son lineales, sin embargo, pueden ser extendidos para modelos no lineales.

Aproximadamente durante la misma época, se unen la teoría de los autómatas celulares con la teoría de juegos para dar lugar a una nueva técnica. Uno de los ejemplos más importantes sobre esta técnica es el juego de la vida de Conway. Este “juego” consiste en una cuadrícula infinita hacia todas las direcciones. Los cuadrados son las células y estas tienen 8 vecinas. La evolución del juego es determinada por el estado inicial. Las transiciones dependen de las células vecinas de tal forma que una célula que está muerta, si tiene tres células vecinas vivas, esta estará viva también en el siguiente turno. Estas células vivas permanecen si solo tienen 2 ó 3 células vecinas vivas.

A finales del siglo XX estos autómatas celulares son liberados de sus casillas convirtiéndose en autónomos (agentes) y ahí es cuando nace la simulación mediante sistemas multiagentes.

2.3. Modelos de Epidemia

Terminología:

- S: Individuos susceptibles.
- I: Individuos infectados.
- R: Individuos recobrados.
- E: Individuos en periodo de incubación (del inglés *Exposed*).

Modelos:

- **SIR.** Es una enfermedad que tiene tres tipos de individuos: Susceptibles, infectados y recobrados. El flujo de transiciones de un grupo a otro es el siguiente:

Ilustración 1 - Modelo Enfermedad SIR

- **SIS**. Esta tipo de enfermedad solo tiene dos clases de individuos: Susceptibles e infectados. El flujo de transiciones entre grupos es:

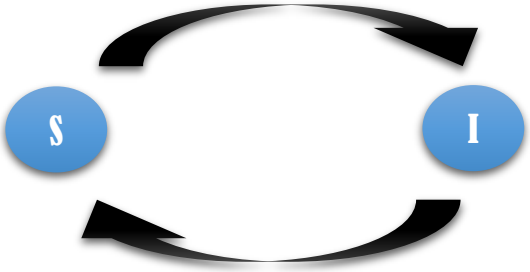

Ilustración 2 - Modelo Enfermedad SIS

- **SEIR**. Es similar al modelo SIR, solo que se le añaden los individuos que están en periodo de incubación. Hay cuatro grupos de individuos: Susceptible, en periodo de incubación (*exposed*), infectados y recobrados. En este caso el flujo de transiciones de un grupo a otro es:

Ilustración 3 - Modelo Enfermedad SEIR

- **SEIS.** Este modelo es muy parecido al SIS, solo que como en el modelo anterior, se le añaden los individuos que están en periodo de incubación.
  En este caso tenemos tres grupos de individuos: Susceptibles, en periodo de incubación (*exposed*) e infectados. El flujo de transiciones entre grupos para este modelo es el siguiente:

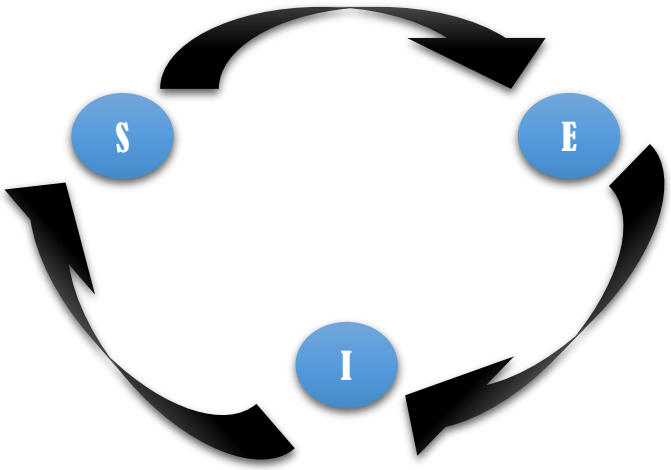

Ilustración 4 - Modelo Enfermedad SEIS

3. Herramientas y Tecnologías

3.1. Comparación de las Diferentes Herramientas y Tecnologías

3.1.1. Bases de Datos

Nuestra aplicación necesitará hacer uso de lectura y escritura en un Sistema Gestor de Bases de Datos. Hemos elegido entre las siguientes (Tabla 1), ya que habíamos trabajado anteriormente con MySQL y Oracle, y PostgreSQL.

	Licencia	OS	Tipo	Rendimiento
MySQL	GLP	Multiplataforma	RDBMS	Alto
PostGreSQL	BSD	Multiplataforma	ORDBMS	Medio
Oracle	Privativa	Multiplataforma	ORDBMS	Alto

Tabla 1 - Sistemas Gestores de Base de Datos.

Al final nos decidimos por MySQL, dado que PostGreSQL tiene un menor rendimiento, mientras que la base de datos de Oracle tiene una licencia de pago. En teoría, es más potente la BD de Oracle pero, como MySQL es bastante potente y tiene una licencia GLP, decidimos que era la mejor opción.

3.1.2. Plataforma de Sistemas Multiagentes

Ya que nuestra aplicación está centrada en agentes, necesitamos un motor de sistemas multiagentes. Hay una cantidad muy grande de plataformas para el desarrollo de sistemas multiagentes. Hemos estado barajando entre las plataformas mostradas en la tabla 2, debido a que todas están basados en java, lenguaje en el que queríamos desarrollar la aplicación.

	Licencia	Comunidad y soporte	FIPA	Compatibilidad	Entorno de Simulación
JADE	LGPL	Muy Alta	Sí	Alta	No
Repast Symphony	AFL	Media	No	Baja	Sí
MASON	BSD	Alta	No	Media	Sí

Tabla 2 - Plataformas de Sistemas Multiagente.

Nos hemos decantado por la utilización de JADE, puesto que tiene un gran soporte y una comunidad muy grande detrás. Un gran punto a favor ha sido la compatibilidad con los Servlets y JSF, y también porque cumple con los estándares FIPA. JADE no tiene un entorno de simulación propio como Repast y MASON, pero en cualquier caso, como no queríamos hacer únicamente una aplicación de escritorio, necesitábamos hacer un entorno de simulación para poder coger los datos y mostrarlos en la aplicación web.

3.2. Herramientas Utilizadas

3.2.1. JADE

JADE (Java Agent DEvelopment Framework) es un framework totalmente implementado en Java y con licencia LGPL para desarrollar aplicaciones basadas en agentes cumpliendo las especificaciones FIPA para sistemas multiagentes interoperables.

Además de suministrar una API para la creación de agentes y un sistema de comunicación entre agentes par a par (Peer to Peer) basada en paso de mensajes asíncronos, también proporciona una interfaz gráfica y un conjunto de herramientas que facilitan la depuración y el control de nuestro sistema durante el desarrollo.

JADE proporciona una plataforma que es donde se despliegan todos los agentes. Cada plataforma puede tener uno o más contenedores. Siempre tiene el contenedor principal, que es donde se encuentra el AMS y DF.

Un agente es un proceso que habita en la plataforma, más concretamente en un contenedor y normalmente ofrece uno o más servicios que pueden publicarse como una descripción del servicio en las “páginas amarillas”. Un Agente tiene un identificador único denominado AID que hace que se pueda distinguir de forma inequívoca. El AID tiene el siguiente formato: <nombre_agente>@<nombre_plataforma>.

Los agentes en JADE se implementan heredando de la clase Agent. Los agentes tienen dos métodos principales: *setup*, que inicializa el agente, y *takeDown*, al que se llama cuando el agente muere. Al agente se le pueden añadir distintos comportamientos (*Behaviour*).

Hay dos tipos de comportamientos, los comportamientos simples y los comportamientos compuestos.

Los comportamientos simples genéricos tienen que extender de la clase **Behaviour** e implementar el los métodos *action*, donde se ejecuta la tarea y *done*, que devuelve *true* cuando finalice el comportamiento, y *false* mientras se siga

ejecutando. También hay algunos comportamientos simples con el método *done* implementado como son ***OneShotBehaviour*** que se ejecuta una sola vez, ya que el método *done* devuelve *true,* y ***CyclicBehaviour*** que el método *done* siempre devuelve *false*, por lo que finaliza cuando el agente muere, es decir, se ejecuta cíclicamente hasta que el agente termina. En ambos casos hay que extender de la clase mencionada e implementar el método *action*. Además, hay una clase que hereda de ***CyclicBehaviour*** denominada ***TickerBehaviour*** que lo que hace es ejercerlo cíclicamente cada cierto periodo de tiempo y hay que implementar la función *onTick*, y decirle cada cuántos milisegundos se repite dicho comportamiento.

El AMS (*Agent Management System*) es componente obligatorio y solo puede existir uno por plataforma. Es un agente que se encarga de gestionar el funcionamiento de la plataforma. Hace tareas como la creación y eliminación de agentes, y la supervisión de la migración de los agentes entre plataformas. Cada agente debe registrarse con el AMS para obtener un AID válido, esta operación en JADE la realizan los agentes automáticamente en el agente AMS por defecto. El AMS es el encargado de mantener el directorio de los identificadores de los agentes y su estado. La vida de un agente termina borrándose del directorio del AMS, es decir, notificando al AMS que va a dejar de existir.

El DF (*Directory Facilitator*) es un componente opcional de la plataforma que presta un servicio de páginas amarillas para los demás agentes. El DF mantiene una lista exacta, completa y actualizada de los servicios prestados por los agentes. Si un agente desea dar a conocer sus servicios debe solicitar al DF que registre la descripción del servicio. Posteriormente los agentes pueden solicitar la cancelación del registro. En cualquier momento un agente podrá solicitar al DF modificar su registro. Además, cualquier agente podrá emitir una solicitud de búsqueda al DF para descubrir descripciones que coincidan con los criterios de búsqueda.

El ciclo de vida de los agentes cumple con el estándar propuesto por FIPA y puede estar en cualquiera de los siguientes estados:

- **Iniciado:** el agente ha sido creado, pero aún no está registrado en el AMS, por lo cual no tiene nombre ni dirección y esto conlleva que todavía no se puede comunicar con otros agentes.
- **Activo:** en este estado el agente ya está registrado en el AMS y tiene un nombre, una dirección. Al tener el AID ya se puede comunicar con otros agentes y puede acceder a todas las características de JADE.
- **Suspendido:** el agente está parado y su hebra está detenida. En este estado el agente no ejecuta ningún comportamiento.
- **En espera:** está bloqueado esperando por algún evento. Su hebra está dormida y se despertará cuando se cuando reciba un mensaje.

- **Desconocido:** el agente ha sido eliminado y su hilo de ejecución ha terminado. También ha sido eliminado del registro del AMS.
- **Tránsito:** un agente móvil está en este estado durante la migración a una nueva localización. El sistema guarda los mensajes en el buffer hasta que el agente vuelve a estar activo, por lo que no se pierde ningún mensaje.

Un punto que hay que destacar es que un sistema desarrollado en JADE puede ser distribuido en distintas máquinas.

3.2.2. JSF

Dado que se pretende hacer una aplicación web en java, hemos decidido utilizar el Framework JSF (*JavaServer Faces*) que funciona sobre JSP. Este framework simplifica mucho el desarrollo de interfaces de usuario y también facilita la separación entre presentación y comportamiento que no se puede conseguir utilizando solo JSP. Además también incorpora componentes de interfaz de usuario, gestión de eventos, validación de datos en el servidor, etc. También define el flujo de navegación entre páginas. Tiene dos librerías de etiquetas propias que permiten hacer una interfaz JSF sobre JSP. Asimismo, incluye soporte para internacionalización y accesibilidad.

Lo más interesante de JSF es división entre presentación y comportamiento tan simple. Por un lado tenemos las *JSF Pages*, son páginas en xhtml que utilizan las librerías de JSF, y por otro lado, los *Managed Beans,* que son clases java que tienen la anotación @ManagedBean. Para poder acceder a los atributos de los Managed Beans estos tiene que tener su getter y setter correspondiente.

El flujo normal que se suele seguir es: primero muestra un formulario web, después se instancia al Bean o los Beans correspondientes, posteriormente se invoca el método controlador indicado anteriormente en el tag action de la etiqueta en la JSF Page, seguidamente el método del action devuelve una condición y se muestra una página como resultado.

4. Análisis

4.1. Metodología de Desarrollo

Actualmente existen dos estándares de metodologías de desarrollo. Por un lado, tenemos la metodología tradicional en cascada, que hace que sea más rígido el proyecto y que el cambio de los requisitos *a posteriori* sea más costoso, y por otro lado, tenemos las metodologías ágiles, que son mucho más flexibles y abiertas a cambios.

Nos hemos decantado por utilizar una metodología ágil, ya que estas están pensadas para el desarrollo en grupo, y como es un proyecto grupal, creemos que esta es la mejor opción. También por razones geográficas de los distintos integrantes del grupo necesitamos trabajar de manera distribuida, y para poder avanzar más rápidamente. Otra razón por la cual hemos elegido este tipo de metodología es porque es difícil predecir inicialmente qué requisitos software se mantendrán y cuáles cambiarán.

Las principales características de las metodologías ágiles es el desarrollo incremental, dividir los requisitos en tareas pequeñas, es decir, la simplicidad y sobre todo la comunicación entre los miembros del equipo.

Nosotros nos hemos decantando por utilizar la metodología Scrum, en la cual se solapan las distintas fases de desarrollo en cada iteración. Cada iteración proporciona un resultado completo, incrementando el producto final. Para seguir esta metodología de trabajo vamos a utilizar la herramienta Trello, que es una especie de tablón de notas donde se colocan las distintas tareas que tiene que realizar cada miembro del grupo, y saber si está sin hacer, en curso, completada sin testear, o finalizada.

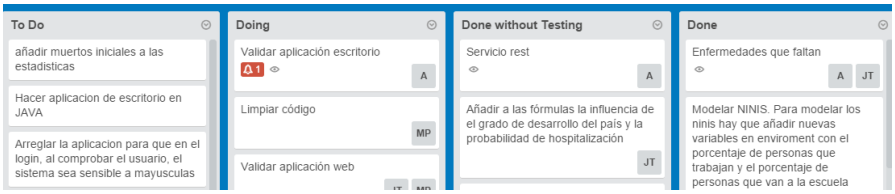

Ilustración 5 - Ejemplo de Uso de Trello

4.2. Actores

En esta aplicación habrá dos actores: el usuario no autenticado y el usuario (ya autenticado), que será el responsable de gestionar los entornos, las simulaciones y las enfermedades.

4.3. Requisitos

4.3.1. Requisitos Funcionales Sistema Multiagente

El sistema multiagente realiza una simulación de una enfermedad en un entorno dinámico. Estas enfermedades afectan a los humanos, que son representados por agentes, que viven en ese entorno. Estos humanos serán creados por el sistema antes de iniciarse la simulación. Para que la simulación cumpla los objetivos del proyecto, se requiere la implementación de ciertas funcionalidades. En la tabla 3 se listan los requisitos funcionales de la simulación.

Requisito	Descripción
Movimiento	El humano se podrá mover de un lugar a otro, dentro de un entorno.
Comunicación	El humano podrá tener comunicación con otros humanos de un mismo lugar.
Lugares frecuentes	El humano podrá tener una lista de lugares frecuentes, entre ellos se encuentran la casa (desde el primer instante) y el trabajo, si el humano trabaja.
Contagio	El humano podrá contagiar a otros humanos que estén en un mismo lugar. El contagio está condicionado por diferentes parámetros, tanto de la enfermedad, como del entorno y del propio humano.
Recuperación	El humano podrá recuperarse de una enfermedad. Esta recuperación está condicionada por diferentes parámetros tanto de la enfermedad, como del entorno y del propio humano.
Incubación	El humano podrá incubar una enfermedad durante un periodo de tiempo. Durante la incubación puede contagiar a otros humanos. Al final de la incubación, el humano enferma. Durante el periodo de incubación el humano no puede contagiar a otros humanos.
Muerte	El humano puede morir a causa de una enfermedad.

Tabla 3 - Requisitos Funcionales Sistema Multiagente

4.3.2. Requisitos Funcionales Aplicaciones

Requisito	Descripción
Acceder a la aplicación	El usuario no autenticado podrá acceder a la aplicación y autenticarse con su nombre de usuario y contraseña.
Creación de entornos	Se podrán crear nuevos entornos para poder hacer distintas simulaciones. Dichos entornos deberán permitir elegir un nombre, el número de habitantes, el área, el nivel de desarrollo, el factor de personas con mayor riesgo, la media de personas por casa, la media de personas por trabajo, el porcentaje de personas que estudian y el porcentaje de personas que trabajan.
Consulta de entornos	Se podrá consultar un listado de los entornos creados por el usuario. Al seleccionar un entorno se visualizará todos los campos asociados a ese entorno.
Edición de entornos	Se podrán modificar todos los campos de un entorno siempre y cuando este entorno no forme parte de una simulación.
Eliminación de entornos	Se podrá eliminar un entorno siempre y cuando no forme parte de una simulación.
Creación de enfermedades	Se podrán crear nuevas enfermedades para poder hacer distintas simulaciones. Dichas enfermedades deben permitir elegir un nombre, la distancia de infección, el tiempo mínimo de infección, el tiempo máximo de infección, el tiempo medio de infección, el ratio de muerte, el ratio de infectividad y el tipo de enfermedad. Si el tipo de enfermedad tiene un periodo de incubación, se deberá introducir el tiempo máximo, el mínimo y el tiempo medio de incubación.
Consulta de enfermedades	Se podrá consultar un listado de las enfermedades. Al seleccionar una enfermedad se visualizarán todos los campos de dicha enfermedad.
Edición de enfermedades	Se podrán modificar todos los campos de una enfermedad siempre y cuando dicha enfermedad no esté asociada a ninguna.
Eliminación de enfermedades	Se podrá eliminar una enfermedad siempre y cuando no esté asociada a una simulación.
Hacer simulaciones	Para llevar a cabo una simulación se debe poder elegir un entorno y una enfermedad entre los distintos que tiene el usuario. También se deberá poder insertar distintos valores de simulación como son el nombre, el número de infectados iniciales, el número de personas en periodo de incubación iniciales, el número de personas muertas iniciales, el nivel de aceptancia (predisposición

	de los individuos de aceptar que están enfermos) y el número de días que durará la simulación.
Consulta de simulaciones	Se podrá ver un listado de todas las simulaciones del usuario. Al seleccionar una simulación se mostrarán distintos datos relevantes de la simulación.
Visualización de gráfica de la simulación	Se podrá visualizar una gráfica con el número de muertos, el número de infectados, el número de expuestos, el número de personas susceptibles y el número de personas recuperadas con respecto al tiempo.
Visualizar la simulación	El usuario podrá visualizar cómo se desarrolla una simulación con respecto al tiempo.
Eliminación de simulaciones	El usuario podrá eliminar cualquiera de sus simulaciones.
Navegación	Se deberá facilitar la navegación entre las distintas secciones (en la aplicación web).

Tabla 4 - Requisitos Funcionales Aplicaciones

4.3.3. Requisitos No Funcionales.

Requisito	**Descripción**
Usabilidad	La aplicación deberá ser lo más intuitiva posible, haciendo que sea cómoda y fácil de usar para cualquier tipo de usuario.
Rendimiento	Se minimizará, en la medida de lo posible, el tiempo de simulación. Para minimizar el tiempo de ejecución se desarrollará de forma que use todos los recursos que tenga la máquina y se ejecute lo más rápido posible.
Mantenimiento	El código deberá estar bien comentado y estructurado, con su correspondiente sangrado, para facilitar su posterior mantenimiento.
Integración	La aplicación web deberá poder integrarse en un servidor de aplicaciones Java EE.
Interoperabilidad	La aplicación web debe poder conectarse con la plataforma de sistemas multiagente Jade.
Documentación	Se incluirá un manual de usuario entendible por usuarios de cualquier nivel.

Portabilidad	Se podrá acceder a la aplicación web desde cualquier navegador web, pero especialmente para Mozilla Firefox, Google Chrome y Safari. La aplicación de escritorio deberá ser multiplataforma.
Almacenamiento	Los datos de los usuarios, como son las enfermedades, los entornos y la simulación, deben ser almacenados en una base de datos.
Interfaz	La interfaz debe ser agradable e intuitiva.

Tabla 5 - Requisitos No Funcionales

4.4. Casos de Uso

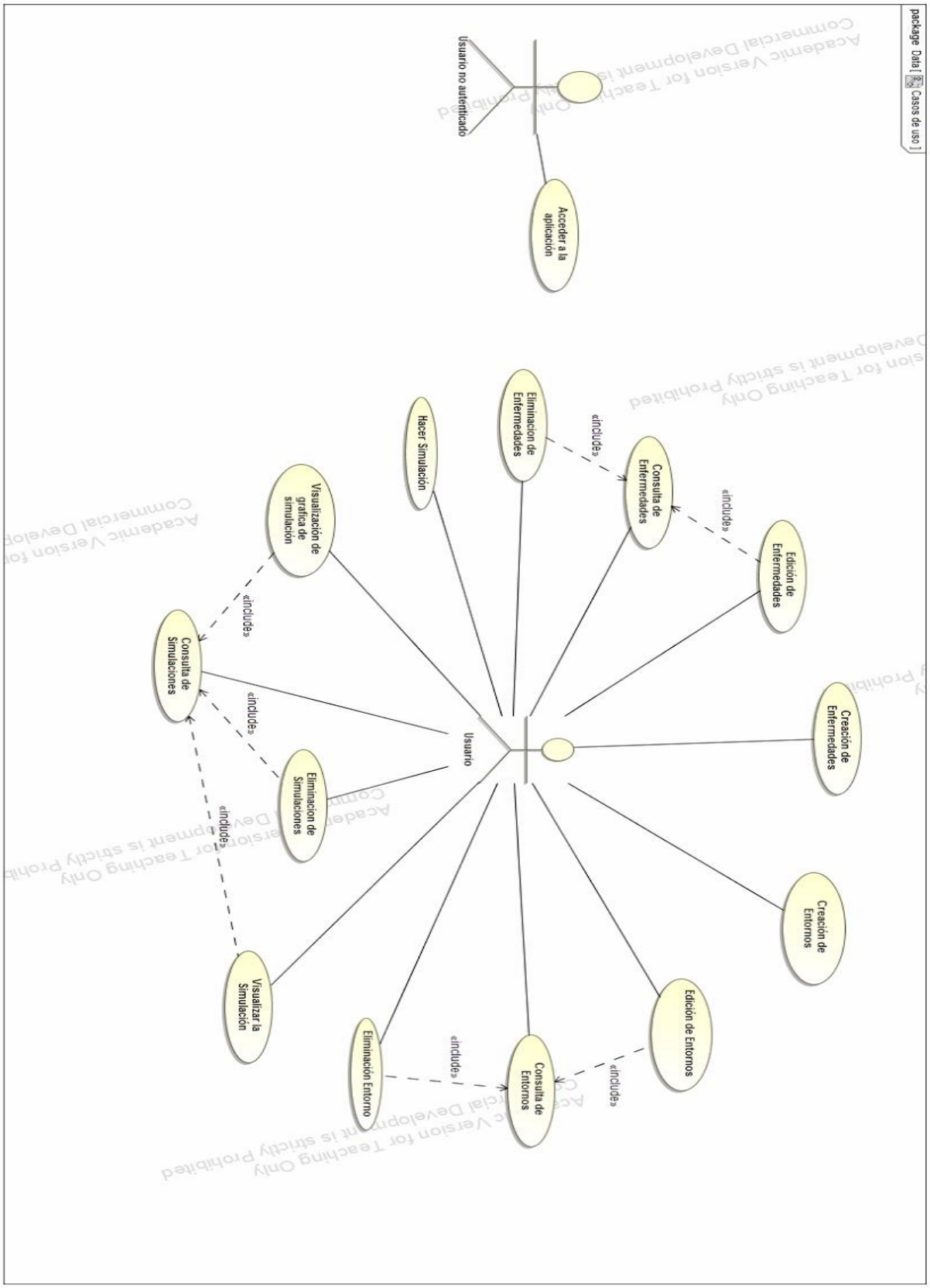

Ilustración 6 - Diagrama de Casos de Uso

Estos casos de usos (Ilustración 6) sirven tanto para la aplicación de escritorio, como para la aplicación web. Concretamente estos han sido creados para la aplicación web, la única diferencia existente sería que la aplicación de escritorio, el caso de uso de consulta de simulaciones, llevaría incluida también la consulta de enfermedades y la consulta de entorno.

La idea principal de la aplicación de escritorio es que se puedan ver todos los datos del entorno, la enfermedad y la simulación en una misma ventana. Entonces, cuando se quiera consultar los datos de una simulación, a su vez se actualizarán todos los datos del entorno y enfermedad asociados a esa simulación en la ventana, lo que facilitará enormemente la consulta de información con respecto a la aplicación web.

4.4.1. Acceder a la Aplicación

Descripción	El usuario no autenticado accede a la aplicación, se identifica y accede a la ventana principal.
Código	CU1
Pre-condición	El usuario tiene que estar registrado.
Post-condición	El usuario está identificado y accede a la ventana principal de la aplicación.
Escenario Principal	1. El usuario introduce el nombre de usuario. 2. El usuario introduce la contraseña. 3. El usuario y la contraseña son correctos. 4. Se muestra la ventana principal.
Escenario Secundario	1. El usuario introduce el nombre de usuario. 2. El usuario introduce la contraseña. 3. El usuario y la contraseña no son correctos. 4. Se muestra un mensaje de error que explique que el usuario y la contraseña no son correctos.
Información Adicional	

Tabla 6 - CU Acceder a la Aplicación

4.4.2. Creación de Entornos

Descripción	Un usuario crea un nuevo entorno.
Código	CU2
Pre-condición	El usuario está identificado.
Post-condición	Se crea un nuevo entorno.

Escenario Principal	1. El usuario va a la página de entornos. 2. El usuario completa todos los datos del formulario (todos son obligatorios): • Nombre • Número de habitantes • Área • Nivel de desarrollo • Porcentaje personas con mayor riesgo • Media de personas por casa • Media de personas por trabajo • Porcentaje de personas que estudian • Porcentaje de personas que trabajan 3. El usuario pulsa el botón crear y todos los datos son correctos. 4. Se inserta el nuevo entorno en la base de datos. 5. En la página de entornos se muestra el nuevo entorno creado.
Escenario Secundario	Los pasos 1 y 2 son similares. 3. El usuario pulsa el botón crear y los datos introducidos no son correctos. 4. Se muestra un mensaje con los errores.
Información Adicional	

Tabla 7 - CU Creación de Entornos

4.4.3. Consulta de Entornos

Descripción	El usuario puede consultar todos los entornos, pudiéndose ver los datos de los mismos.
Código	CU3
Pre-condición	El usuario está identificado.
Post-condición	Se muestra una lista de los entornos creados por el usuario y se visualizan los datos del entorno seleccionado.
Escenario Principal	1. El usuario accede a la ventana de entornos. 2. Se muestra una lista con los entornos del usuario. 3. El usuario selecciona un entorno. 4. Se muestran los datos referentes a ese entorno.
Escenario Secundario	1. El usuario va a la ventana de entornos. 2. No hay ningún entorno, por lo que la lista está vacía.

Tabla 8 - CU Consulta de Entornos

4.4.4. Edición de Entornos

Descripción	El usuario edita los valores de uno de sus entornos.
Código	CU4
Pre-condición	El usuario está identificado. El entorno no es utilizado en ninguna simulación.
Post-condición	Se guardan los nuevos valores del entorno.
Escenario Principal	1. El usuario consulta el entorno (CU3). 2. El usuario edita los campos correctamente. 3. El usuario pulsa el botón de editar y se guardan los nuevos datos del entorno.
Escenario Secundario	1. El usuario consulta el entorno (CU3). 2. El usuario introduce en los campos parámetros incorrectos. 3. El usuario pulsa el botón de editar y se muestra un mensaje de error informando de los campos incorrectos.
Información Adicional	

Tabla 9 - CU Edición de Entornos

4.4.5. Eliminación de Entornos

Descripción	El usuario elimina un entorno.
Código	CU5
Pre-condición	El usuario está identificado. El entorno no es utilizado en ninguna simulación.
Post-condición	El entorno seleccionado es eliminado.
Escenario Principal	1. El usuario consulta el entorno (CU3). 2. El usuario pulsa el botón de eliminar. 3. El entorno se elimina correctamente.
Escenario Secundario	1. El usuario consulta el entorno (CU3). 2. El usuario pulsa el botón de eliminar. 3. Como el entorno es utilizado en una simulación, se muestra el mensaje de error correspondiente.
Información Adicional	

Tabla 10 - CU Eliminación de Entornos

4.4.6. Creación de Enfermedades

Descripción	El usuario crea una nueva enfermedad.
Código	CU6
Pre-condición	El usuario está identificado.
Post-condición	Se crea una nueva simulación.
Escenario Principal	1. El usuario va a la página de entornos. 2. El usuario completa todos los datos del formulario: • Nombre (obligatorio) • Distancia de infección (obligatorio) • Tipo de enfermedad (obligatorio): puede ser de tipo SIR, SIS, SEIS y SEIR • Tiempo medio de infección (obligatorio) • Tiempo máximo de infección (obligatorio) • Tiempo mínimo de infección (obligatorio) • Tiempo medio de incubación (obligatorio si la enfermedad es tipo SEIR o SEIS) • Tiempo máximo de incubación (obligatorio si la enfermedad es tipo SEIR o SEIS) • Tiempo mínimo de incubación (obligatorio si la enfermedad es tipo SEIR o SEIS) • Ratio de muerte (obligatorio) • Ratio de infectividad (obligatorio) 3. El usuario pulsa el botón crear y todos los datos son correctos. 4. Se inserta la nueva enfermedad en la base de datos. 5. En la página de enfermedades se muestra la nueva enfermedad creada.
Escenario Secundario	Los pasos 1 y 2 son similares. 3. El usuario pulsa el botón crear y los datos introducidos no son correctos. 4. Se muestra un mensaje informado de los errores.
Información Adicional	Para crear una enfermedad de tipo SIR O SIS, no se mostrarán los campos g, h e i, en cambio, si la enfermedad es de tipo SEIS o SEIR estos campos se muestran y son obligatorios.

Tabla 11 - CU Creación de Enfermedades

4.4.7. Consulta de Enfermedades

Descripción	El usuario puede consultar todas las enfermedades, pudiéndose ver los datos de las mismas.
Código	CU7
Pre-condición	El usuario está identificado.
Post-condición	Se muestra una lista de las enfermedades creadas por el usuario y se visualizan los datos de la enfermedad seleccionada.
Escenario Principal	1. El usuario accede a la ventana de enfermedades. 2. Se muestra una lista con las enfermedades del usuario. 3. El usuario selecciona una enfermedad. 4. Se muestran los datos referentes a esa enfermedad.
Escenario Secundario	1. El usuario accede a la ventana de enfermedades. 2. No hay ninguna enfermedad por lo que la lista está vacía.
Información Adicional	Al mostrar los datos enfermedad de tipo SIR O SIS, no se mostrarán los campos g, h e i; en cambio, si la enfermedad es de tipo SEIS o SEIR estos campos se muestran.

Tabla 12 - CU Consulta de Enfermedades

4.4.8. Edición de Enfermedades

Descripción	El usuario edita los valores de una de sus enfermedades.
Código	CU8
Pre-condición	El usuario está identificado. La enfermedad no es utilizada en ninguna simulación.
Post-condición	Se guardan los nuevos valores de la enfermedad.
Escenario Principal	1. El usuario consulta la enfermedad (CU7). 2. El usuario edita los campos correctamente. 3. El pulsa el botón editar y se guardan los nuevos datos de la enfermedad.
Escenario Secundario	1. El usuario consulta la enfermedad (CU7).

	2. El usuario introduce en los campos parámetros incorrectos. 3. El usuario pulsa el botón editar y se muestra un mensaje de error informando de los campos incorrectos.
Información Adicional	Para editar una enfermedad de tipo SIR O SIS, no se mostrarán los campos g, h e i; en cambio, si la enfermedad es de tipo SEIS o SEIR estos campos se muestran y son obligatorios. Si se cambia el tipo de enfermedad deben aparecer los campos correctos. Todos los campos que aparecen en la vista son obligatorios.

Tabla 13 - CU Edición de Enfermedades

4.4.9. Eliminación de Enfermedades

Descripción	El usuario elimina una enfermedad.
Código	CU9
Pre-condición	El usuario está identificado. La enfermedad no es utilizada en ninguna simulación.
Post-condición	La enfermedad seleccionada es eliminada.
Escenario Principal	1. El usuario consulta la enfermedad (CU7). 2. El usuario pulsa el botón de eliminar. 3. La enfermedad se elimina correctamente.
Escenario Secundario	1. El usuario consulta la enfermedad (CU7). 2. El usuario pulsa el botón de eliminar. 3. Como la enfermedad es utilizada en una simulación, se muestra el mensaje de error correspondiente.
Información Adicional	

Tabla 14 - CU Eliminación de Enfermedades

4.4.10. Hacer Simulación

Descripción	El usuario hace una simulación.
Código	CU10
Pre-condición	El usuario está identificado. Debe haber una enfermedad y un entorno creados.
Post-condición	Los datos de la simulación están almacenados.

Escenario Principal	1. El usuario selecciona un entorno de la lista de entornos. 2. El usuario selecciona una enfermedad de la lista de enfermedades. 3. Introduce los parámetros de simulación: (todos los campos son obligatorios) • Nombre • Número de días • Número de infectados iniciales • Número de personas en periodo de incubación iniciales • Número de muertos iniciales • Aceptancia (predisposición de los individuos de aceptar que están enfermos) 4. El usuario pulsa el botón simular, los parámetros son correctos y comienza la simulación.
Escenario Secundario	Los pasos 1, 2 y 3 son los mismos que en el escenario principal. 1. El usuario pulsa el botón simular y los parámetros no son correctos. 2. Se muestra un mensaje de error indicando cuáles son los parámetros incorrectos.
Información Adicional	El número de personas en periodo de incubación solo se mostrará y será obligatorio introducirlo si la enfermedad es de tipo SEIS o SEIR.

Tabla 15 - CU Hacer Simulación

4.4.11. Consulta de Simulaciones

Descripción	El usuario puede ver un listado con todas las simulaciones, pudiéndose ver los datos de las mismas.
Código	CU11
Pre-condición	El usuario está identificado.
Post-condición	Se muestra una lista de las simulaciones hechas por el usuario y se visualizan los datos de la simulación seleccionada.
Escenario Principal	1. El usuario va a la ventana de simulaciones. 2. Se muestra una lista con las simulaciones del usuario. 3. El usuario selecciona una simulación. 4. Se muestran los datos referentes a esa simulación.

Escenario Secundario	1. El usuario accede a la ventana de simulaciones. 2. No hay ninguna simulación, por lo que la lista está vacía.
Información Adicional	

Tabla 16 - CU Consulta de Simulaciones

4.4.12. Visualizar Gráfica de Simulación

Descripción	El usuario puede visualizar en forma de gráfica algunos valores de la simulación.
Código	CU12
Pre-condición	El usuario está identificado. Existe alguna simulación.
Post-condición	Se muestra una gráfica con los valores de la simulación seleccionada.
Escenario Principal	1. El usuario consulta una simulación (CU11). 2. El usuario pulsa el botón ver gráfica. 3. Se muestra una gráfica de la simulación seleccionada.
Escenario Secundario	1. El usuario consulta las simulaciones (CU11). 2. No existe ninguna simulación por lo que no se puede mostrar ninguna gráfica.
Información Adicional	

Tabla 17 - CU Visualizar Gráfica de Simulación

4.4.13. Visualizar Simulación

Descripción	El usuario puede visualizar gráficamente la evolución de la simulación, es decir, el movimiento de los agentes por el entorno con respecto al tiempo.
Código	CU13
Pre-condición	El usuario está identificado. Existe alguna simulación.
Post-condición	Se muestra gráficamente la simulación.
Escenario Principal	1. El usuario consulta una simulación (CU11). 2. El usuario pulsa el botón ver simulación. 3. Se muestra la simulación seleccionada.
Escenario Secundario	1. El usuario consulta las simulaciones (CU11).

	2. No existe ninguna simulación, por lo que no se puede mostrar.
Información Adicional	

Tabla 18 - CU Visualizar Simulación

4.4.14. Eliminación de Simulaciones

Descripción	El usuario elimina una.
Código	CU14
Pre-condición	El usuario está identificado. Existe alguna simulación.
Post-condición	La simulación seleccionada es eliminada.
Escenario Principal	1. El usuario consulta una simulación (CU11). 2. El usuario pulsa el botón de eliminar. 3. La simulación se elimina correctamente.
Escenario Secundario	
Información Adicional	

Tabla 19 - CU Eliminación de Simulaciones

4.5. Análisis de la Base de Datos

En un principio se identificaron 6 entidades en la base de datos:

- **disease:** se almacenarán los distintos datos de cada una de las enfermedades.
- **typedisease:** se almacenarán los distintos tipos de enfermedades que puede haber, en nuestro caso serán *SIR*, *SIS*, *SEIR* y *SEIS*.
- **environment:** se almacenarán los distintos datos de cada uno de los entornos.
- **leveldevelopment:** en esta se almacenarán los distintos niveles de desarrollo que podrá tener un entorno, en este caso serán *low*, *medium*, *high* y *very high.*
- **simulation:** se almacenarán los datos de cada simulación. Cada simulación tendrá asociada un entorno (*environment*) y una enfermedad (*disease*).
- **user:** en esta entidad se almacenarán los distintos usuarios que hay en el sistema.

Posteriormente se decidió añadir dos entidades más para descargar la entidad simulation. En estas entidades se almacenarán datos relacionados con cada instante en la simulación.

- **humaninstant:** se almacenarán los datos de los humanos de la simulación en cada instante.
- **graphicsdata:** en esta entidad se guardarán los datos estadísticos de los distintos instantes de la simulación.

4.6. Interfaz de Usuario

- **Aspecto gráfico**: para la aplicación web se utilizarán el framework JSF y con la librería PrimeFaces para darle un aspecto más elegante y añadirle diversos gráficos más fácilmente. Para la aplicación de escritorio se utilizarán los elementos gráficos que tiene java por defecto y para dibujar los distintos gráficos se usará la librería jFreeChart.
- **CSS**: en la aplicación web se definirá un estilo que sea similar a los objetos de PrimeFaces para que esté todo en armonía y posicionar los distintos elementos adecuadamente.
- **AJAX**: En la aplicación web se utilizará AJAX para hacer peticiones asíncronas y para no tener que recargar toda la página cada vez que el usuario realice alguna acción.
- **Navegación**: En la aplicación de escritorio estará todo integrado en la misma ventana, excepto el acceso del usuario y las distintas gráficas. En la aplicación web habrá un menú lateral donde se podrá navegar entre distintas páginas.

5. Diseño

5.1. Diagrama de Distribución

En este diagrama podemos observar los distintos componentes que forman parte del sistema a alto nivel.

Ilustración 7 - Diagrama de Distribución

Como nodos clientes tenemos el navegador web, que se comunica bidireccionalmente con la aplicación a través del servidor mediante llamadas HTTP, y la aplicación de escritorio que se comunica igualmente en ambas direcciones, pero este se comunica mediante servicios web RESTful.

El servidor de aplicaciones se comunica bidireccionalmente, tanto con la plataforma de simulación multiagentes, como con el servidor de base de datos. Con el servidor de base de datos se comunica mediante JPA (Java Persistence API) y con JADE se comunica mediante un agente Gateway.

JADE también se comunica en ambas direcciones con el servidor de base de datos mediante JDBC.

Para terminar, el servidor de base de datos se comunica bidireccionalmente con la base de datos.

5.2. Diagramas de Clases

5.2.1. Sistema Multiagente de Simulación

A continuación se mostrará un diagrama de clases del sistema Multiagente de simulación (adjunto también en versión digital) para entender bien su estructura. Vamos a empezar a describir el sistema por la clase **AgentGateway,** que es la encargada comunicarse con el servidor e iniciar el sistema, y está relacionada con la clase **Parameters,** que es la se utiliza para pasarle todos los datos a **AgentGateway. MainController**, también está relacionado con **Parameters,** ya que **AgentGateway** le pasa la instancia de este a **MainController.**

MainController está relacionado también con **Simulation**, **Environment** y **Disease. MainControllerBehaviour** es la clase que modela el comportamiento de **MainController,** por lo que está vinculado con este, al igual que ocurre con **SecondaryController** y **SecondaryControllerBehavior**,

Simulation se utiliza principalmente para modelar el tiempo de la simulación, **Disease** modela la enfermedad y **Environment** modela el entorno, y por eso este tiene un conjunto de posiciones **Position** (*world*).

Tanto la clase **LivingBeing, MainController** como **SecondaryController** heredan de la clase **Agent**, que es de JADE, por lo que son agentes. **Human** también es agente, ya que este a su vez hereda de **LivingBeing**.

Un ser vivo (**LivingBeing**) está siempre en una posición (**Position**). Un humano (**Human**) tiene también distintas posiciones almacenadas, de ahí las relaciones con **Position** que son *work*, *frequentPlaces* y *house*, que representan el trabajo, un conjunto de lugares frecuentes y la casa respectivamente.

Human también está relacionado con el entorno por el que se mueve (**Environment**) y con la enfermedad (**Disease**). Dado que **Human** es un agente, también tiene un comportamiento que lo modela la clase **HumanBehaviour,** que tiene una instancia de este.

Las clases **HumanStatus** y **SimulationStatus** se utilizan para almacenar datos durante la simulación. **State**, **Level**, **Risk** y **TypeDisease** son enumerados que modelan el estado de salud de un humano, el nivel de desarrollo, el riesgo que tiene el humano y el tipo de enfermedad respectivamente.

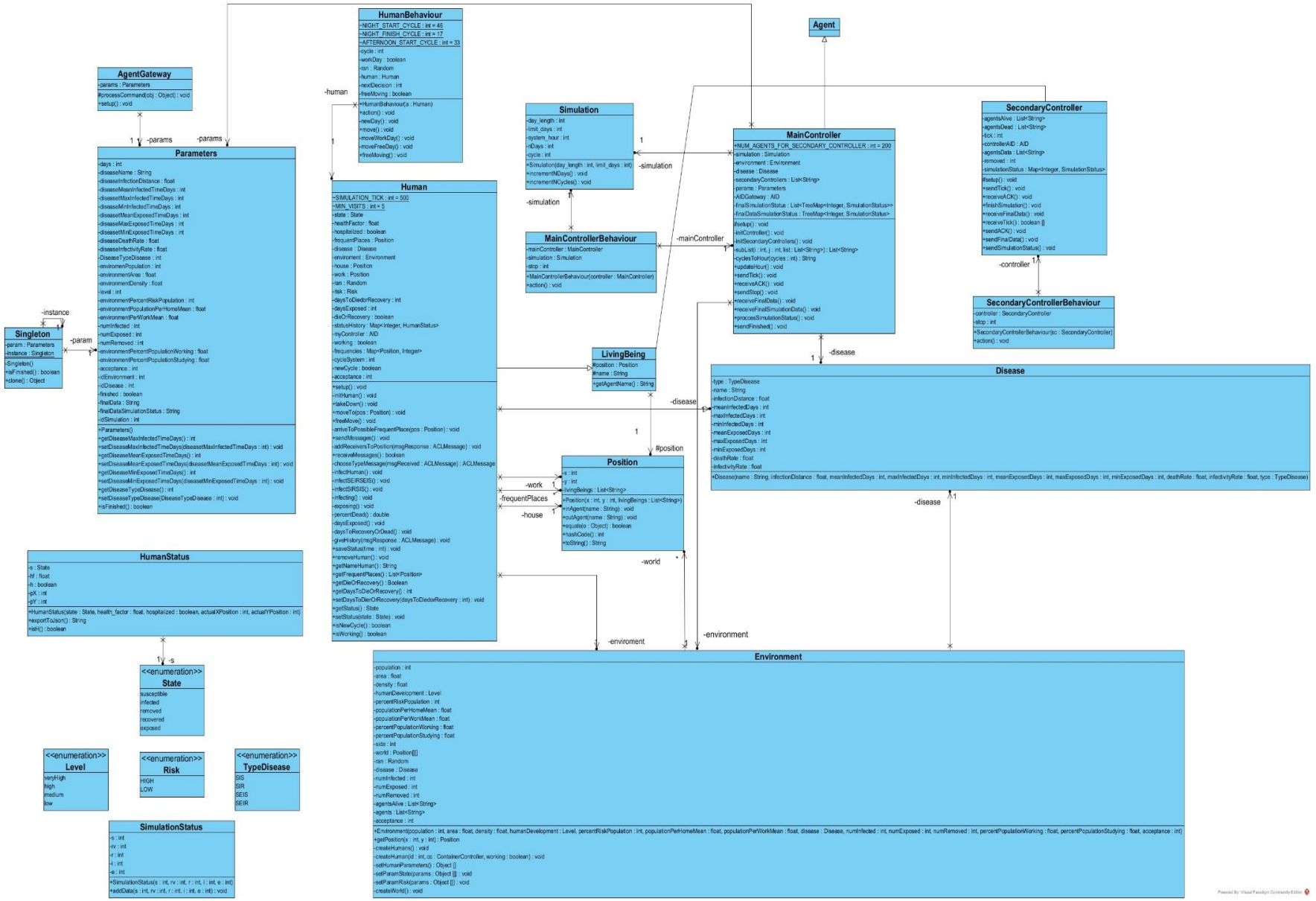

Ilustración 8 - Diagrama de Clases del Sistema Multiagente de Simulación

5.2.2. Aplicación Web

A continuación se mostrarán dos diagramas, solo con el núcleo de las clases de la aplicación web, debido a que son muy grandes. Se podrá ver con más detalle en la versión digital adjunta.

En este primero (ilustración 9) se muestran los servicios RESTful que heredan todos de **AbstractFacade**. Todas estas clases implementan los servicios RESTful que posteriormente serán utilizados en la aplicación de escritorio.

También se muestra la clase **AgentGateway** que se utiliza para la conexión con JADE, que utiliza la clase **Parameters,** como he explicado antes para pasarle los parámetros adecuados con el fin de iniciar la conexión. Como queremos que solo haya una simulación en curso, se ha hecho una clase **Singleton,** que implementa el patrón singleton y que tiene una instancia de la clase **Parameters**.

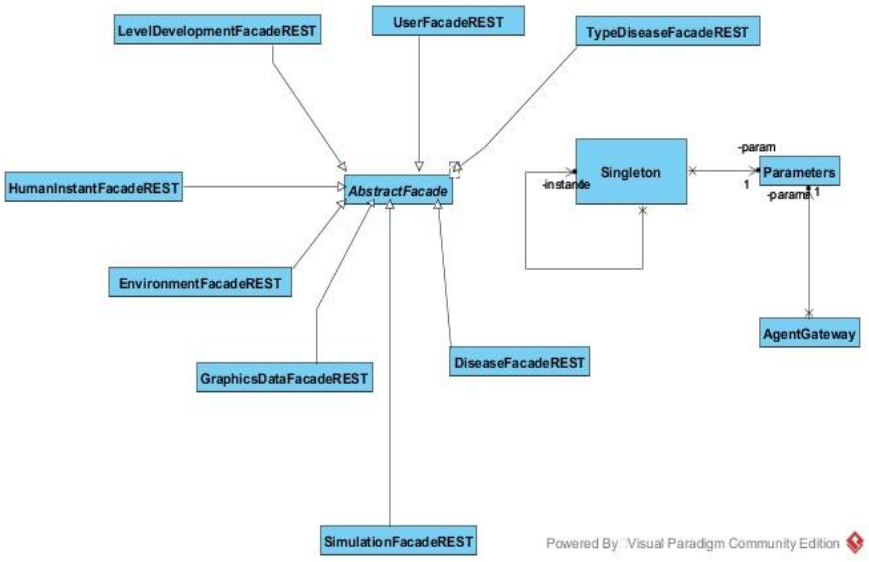

Ilustración 9 - Diagrama de Clases de la Aplicación Web 1

Vamos a comentar el segundo diagrama por filas para su mejor entendimiento.

- En la primera fila podemos ver **AbstractFacade** que utilizan las demás, que tiene los métodos básicos para acceder a la base de datos (*create, edit, remove, find, findAll, findRange* y *count*).
- En la segunda fila vemos todas las Facade que extienden de AbstractFacade, que tiene todos los métodos de **AbstractFacade**, además de los especificados en las correspondientes interfaces implementadas de la tercera fila.
- En la tercera fila tenemos las FacadeLocal que son las interfaces por las que se accede a las Facade. Cada una corresponde a una Facade come se puede observar en el diagrama.
- En la cuarta fila tenemos los ManagedBeans, que son los controladores encargados de las vistas, que utilizan las interfaces locales de las Facade para el acceso a la base de datos.

- En la <u>última fila</u> podemos observar las clases que representan las entidades de la base de datos (LevelDevelopment pertenece a esta última fila).

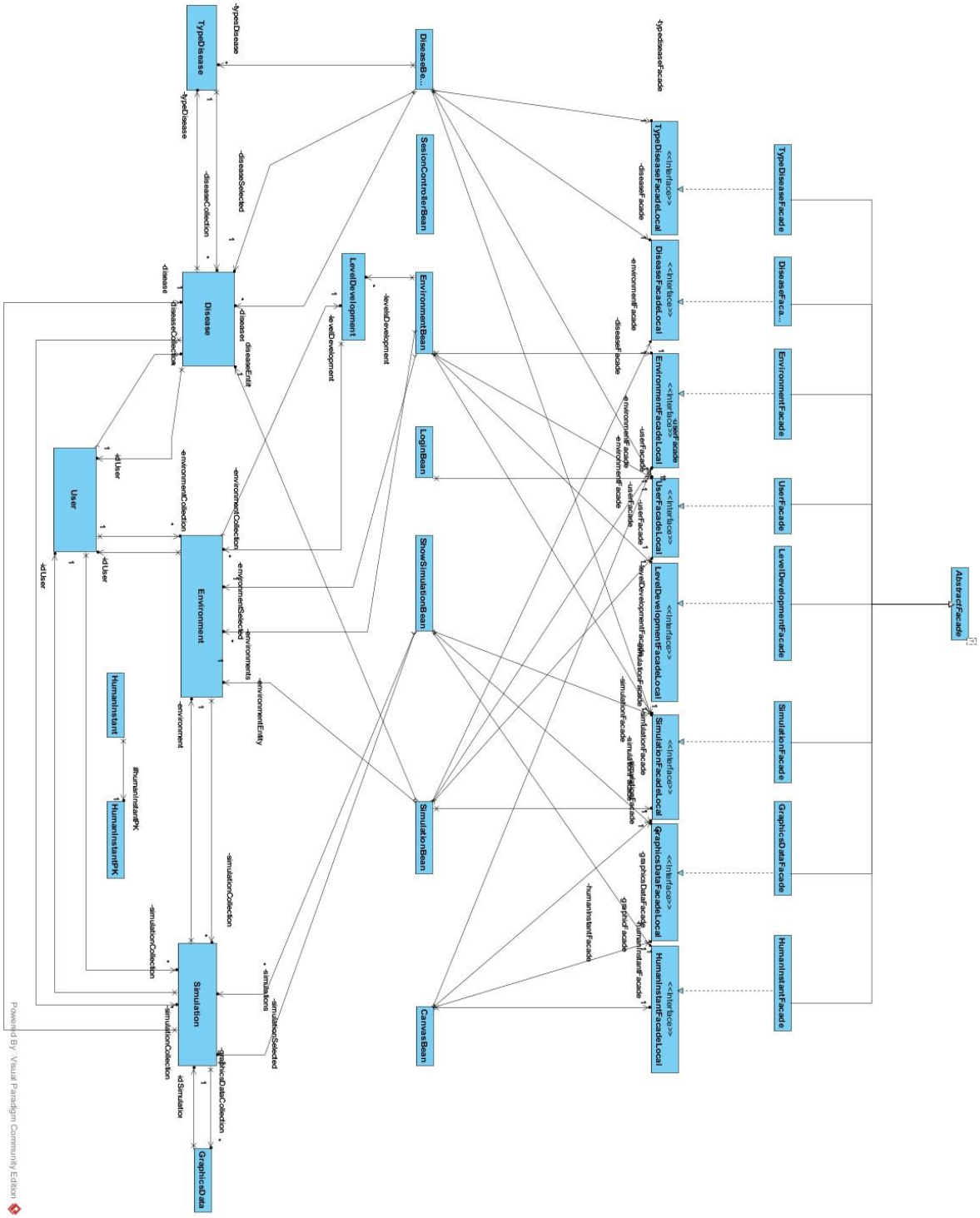

Ilustración 10 - Diagrama de Clases de la Aplicación Web 2

5.2.3. Aplicación de Escritorio

Como en el apartado anterior, se mostrarán los diagramas solo con el núcleo de las clases de la aplicación de escritorio, debido a que es muy grande. Se podrá ver con más detalle en la versión digital adjunta.

Primero, vamos a comentar las clases inconexas del diagrama. Por un lado, tenemos las clases que consumen los servicios REST, que son todas aquellas clases que tienen como sufijo JerseyClient. No están conectados con ninguna otra clase porque solamente tienen distintos métodos abstractos que son llamados desde otras clases, más concretamente desde la clase **Controller** y desde los distintos workers que comentaremos más adelante. Por otro lado, tenemos la clase **MessageError**, una clase auxiliar que se utiliza para mostrar algunos mensajes de error, y la clase **Cliente**, que contiene el método main de la aplicación que lanza el controlador (**Controller**) encargado de todo lo demás.

Las clases **Disease**, **Environment**, **GraphicsData**, **HumanInstant**, **HumanInstantPK**, **LevelDevelopment**, **Simulation**, **TypeDisease** y **User** son las clases que representan las entidades de la base de datos. Estas clases son las mismas que en la aplicación web. Aquí son necesarias para poder consumir los servicios RESTful.

La clase **Controller** es la encargada de controlar todas las vistas: **LoginView**, **SimulationView**, **ShowSimulationView** y **ChartSimulation.**

LoginView es la vista para autenticarse. **SimulationView** es la vista principal donde se muestran todos los datos de las simulaciones, entornos y enfermedades, además también se pueden iniciar simulaciones. **ShowSimulationView** es la vista donde se puede ver la representación gráfica de la simulación. **ChartSimulation** es la vista donde se muestran los gráficos de las simulaciones.

Las clases **WorkerDeleteSimulation**, **WorkerDoSimulation** y **WorkerShowSimulation** heredan de la clase de java SwingWorker. Estas clases son hebras distintas que se encargan de ejecutar código en segundo plano para no dejar paralizada la hebra principal de la GUI.

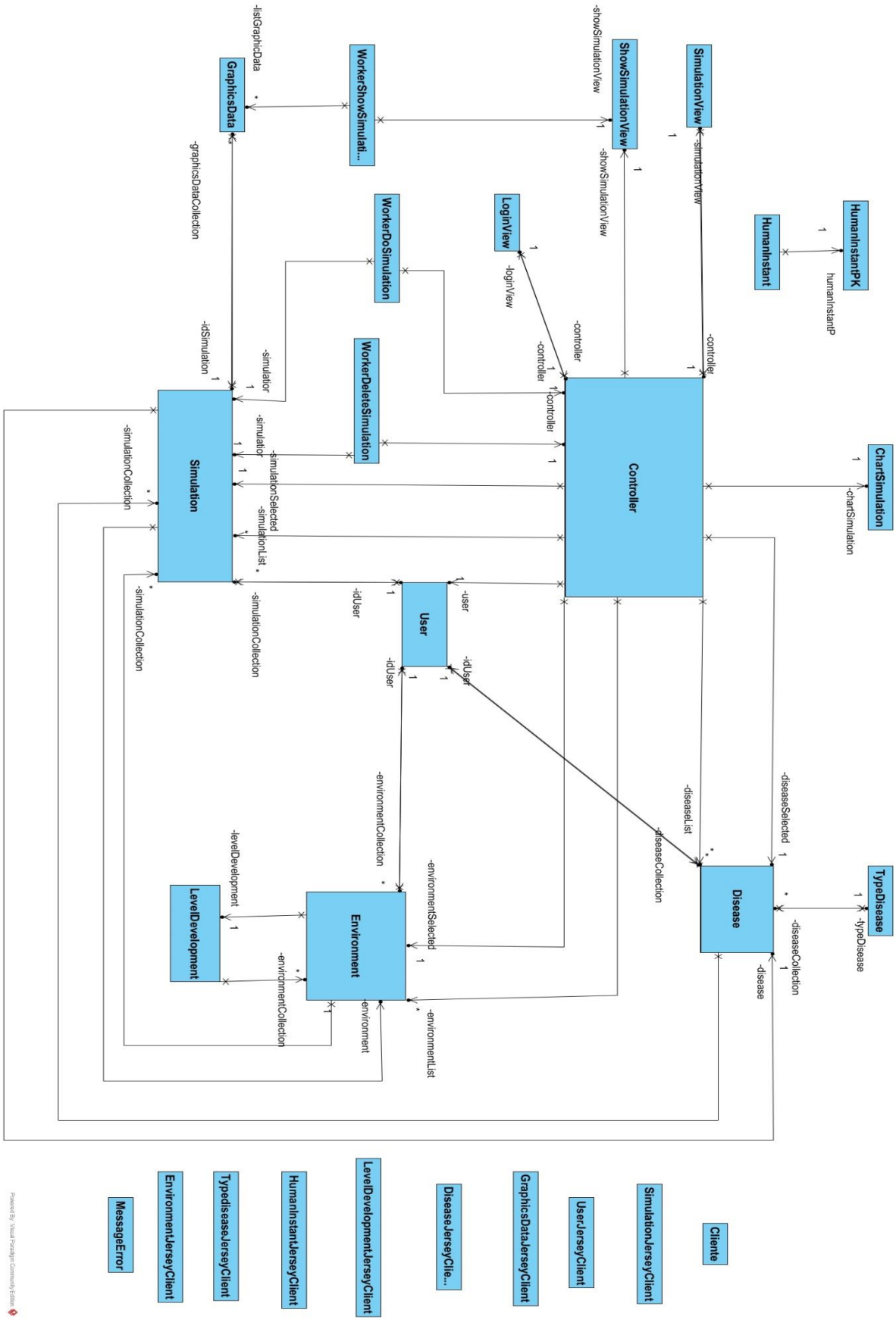

Ilustración 11 - Diagrama de Clases de la Aplicación Escritorio

5.3. Modelo Relacional de la Base de Datos.

Para almacenar los datos tanto de las enfermedades, de los entornos, como de las simulaciones realizadas, se utiliza el siguiente modelo relacional:

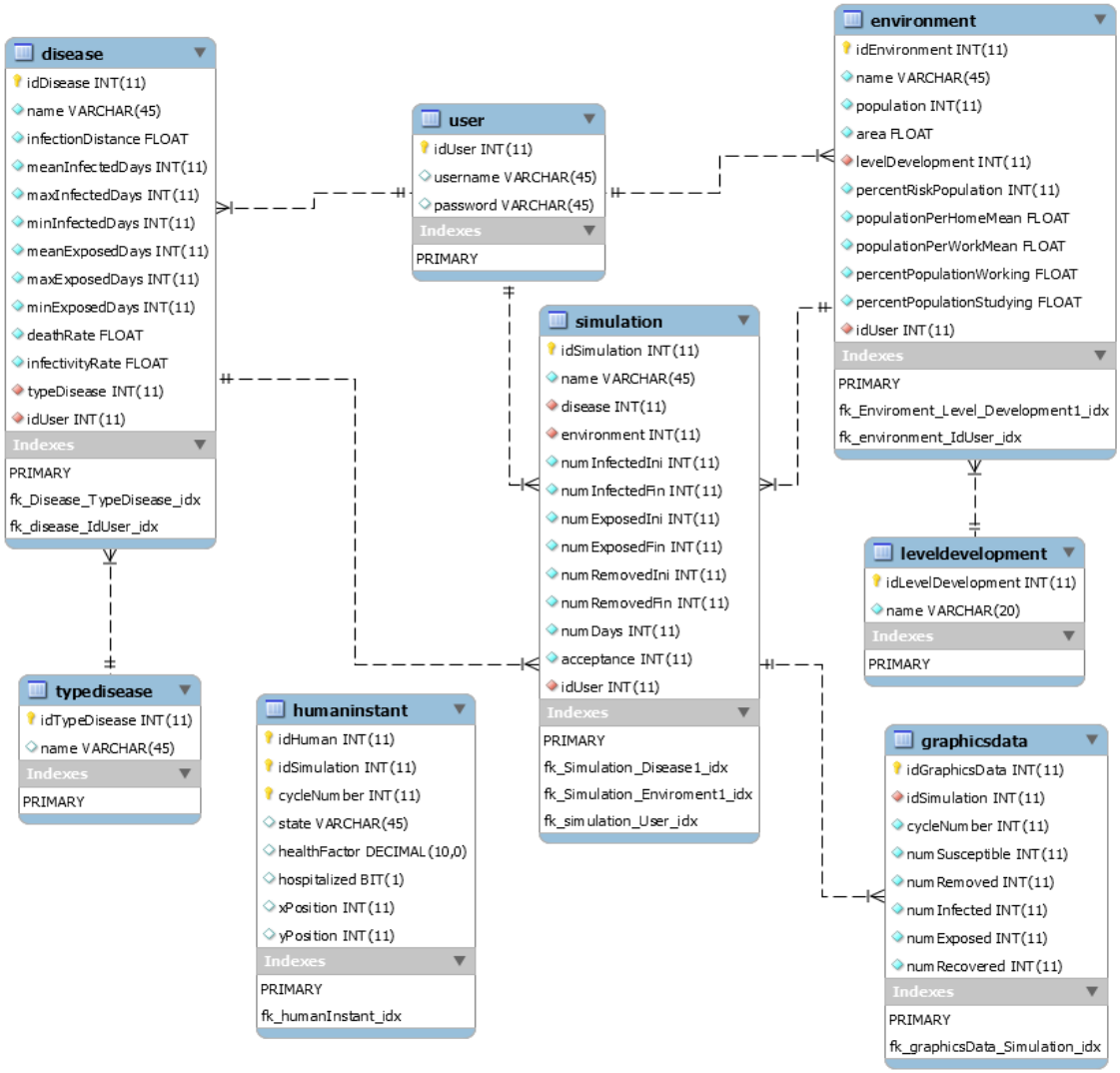

Ilustración 12 - Modelo Relacional de la Base de Datos

- **User:** almacena los distintos usuarios.
  - idUser: clave primaria que identifica a un usuario.
  - username: nombre de usuario.
  - password: contraseña del usuario.
- **simulation:** almacena las distintas simulaciones de los usuarios.
  - idSimulation: clave primaria que identifica a la simulación.
  - name: nombre de la simulación.
  - disease: clave foránea a la clave primaria de la tabla disease asignada por el índice fk_Simulation_Disease1_idx.

    - environment: clave foránea a la clave primaria de la tabla environment asignada por el índice fk_Simulation_Environmente1_idx.
    - idUser: clave foránea a la clave primaria de la tabla disease asignada por el índice fk_Simulation_User_idx.
    - numInfectedIni: número inicial de infectados.
    - numInfectedFin: número final de infectados.
    - numExposedIni: número inicial de personas en periodo de incubación.
    - numExposedIni: número final de personas en periodo de incubación.
    - numRemovedIni: número de muertos iniciales.
    - numRemovedIni: número de muertos finales.
    - numDays: número de días que dura la simulación.
    - acceptance: porcentaje de que una persona trabajadora vaya a trabajar si se encuentra enferma.
- **disease**: almacena las distintas enfermedades de los usuarios.
    - idDisease: clave primaria que identifica la enfermedad.
    - name: nombre de la enfermedad.
    - infectionDistance: distancia de contagio de la enfermedad.
    - meanInfectedDays: número de días medio de infección.
    - maxInfectedDays: número de días máximo de infección.
    - minInfectedDays: número de días mínimo de infección.
    - meanExposedDays: número de días medio de incubación.
    - maxExposedDays: número de días máximo de incubación.
    - minExposedDays: número de días mínimo de incubación.
    - deathRate: ratio de muerte de la enfermedad.
    - infectivityRate: ratio de infectividad de la enfermedad.
    - typeDisease: clave foránea a la clave primaria de la tabla typedisease asignada por el índice fk_Simulation_TypeDisease_idx.
    - idUser: clave foránea a la clave primaria de la tabla disease asignada por el índice fk_disease_IdUser_idx.
- **typedisease**: almacena los distintos tipos de enfermedades.
    - idTypeDisease: clave primaria que identifica los tipos de enfermedad.
    - name: nombre del tipo de enfermedad.
- **environment:** Almacena los entornos creados por los usuarios.
    - idEnvironment: clave primaria que identifica al entorno.
    - name: nombre del entorno.
    - population: número de personas que viven en ese entorno.
    - area: el área que tiene el entorno.
    - levelDevelopment: clave foránea a la clave primaria de la tabla leveldevelopment asignada por el índice fk_Environment_Level_Development1_idx.
    - percentRiskPopulation: porcentaje de población de riesgo.
    - populationPerHomeMean: número media de personas por casa.
    - populationPerWorkMean: número medio de personas por trabajo.

    - percentPopulationWorking: porcentaje de personas que trabajan.
    - percentPopulationStudying: porcentaje de personas que estudian.
    - idUser: clave foránea a la clave primaria de la tabla disease asignada por el índice fk_Simulation_User_idx.
- **leveldevelopment:** almacenara los distintos niveles de desarrollo.
    - idLevelDevelopment**:** clave primaria que identifica el nivel de desarrollo.
    - name**:** nombre del nivel de desarrollo.
- **graphicsdata:** almacena los datos estadísticos de cada instante de una simulación, para poder mostrar gráficas y diversos datos.
    - idGraphicsData: clave primaria que identifica los datos gráficos.
    - idSimulation: clave foráneas a la clave primaria de la tabla simulation asignada por el índice fk_graphicsData_Simulation_idx.
    - cycleNumber: número de ciclo de los datos.
    - numSusceptible: número de personas susceptibles en este ciclo.
    - numRemoved: número de personas muertas en este ciclo.
    - numInfected: número de personas infectadas en este ciclo.
    - numExposed: número de personas en periodo de incubación en este ciclo.
    - numRecovered: número de personas recuperadas en este ciclo.
- **humaninstant:** almacena los datos de cada humano en cada instante de cada simulación, es decir, el estado del humano en cada momento. Esta tabla en un principio tenía una clave foránea hacia la simulación, pero por temas de rendimiento se eliminó. Como cabe imaginar, al tener una entrada por cada instante, humano y simulación, para una simulación se añaden muchísimas entradas en la base de datos. La clave foránea apuntando a la simulación demoraba en exceso el tiempo de introducción y eliminación de las entradas pero, al eliminarla, la inserción y borrado se hace prácticamente de manera instantánea.
    - idHuman: identificador de humano. Forma parte de la clave primaria.
    - idSimulation: identificador de la simulación a la que corresponden estos datos. Forma parte de la clave primaria.
    - cycleNumber: el número de ciclo. Forma parte de la clave primaria.
    - state: estado de salud en el que se encuentra el humano.
    - healthFactor: porcentaje de salud del humano.
    - hospitalized: indica si el humano está hospitalizado en ese instante.
    - xPosition: posición x del humano en el entorno, en ese instante.
    - yPosition: posición y del humano en el entorno, en ese instante.

6. Tareas Realizadas e Implementación

6.1. Sistema de Control de Versiones.

El sistema de control de versiones empleado por todo el grupo ha sido Git. Para ello hemos estado utilizando el servicio Bitbucket, que nos ofrece una plataforma web. Hemos decidido utilizar este servicio aparte de su gratuidad, porque también todos nosotros estábamos familiarizados con el uso de este servicio.

6.2. Estructura del Proyecto

Vamos a empezar describiendo la estructura del proyecto de la aplicación servidora que se puede ver en la ilustración 13.

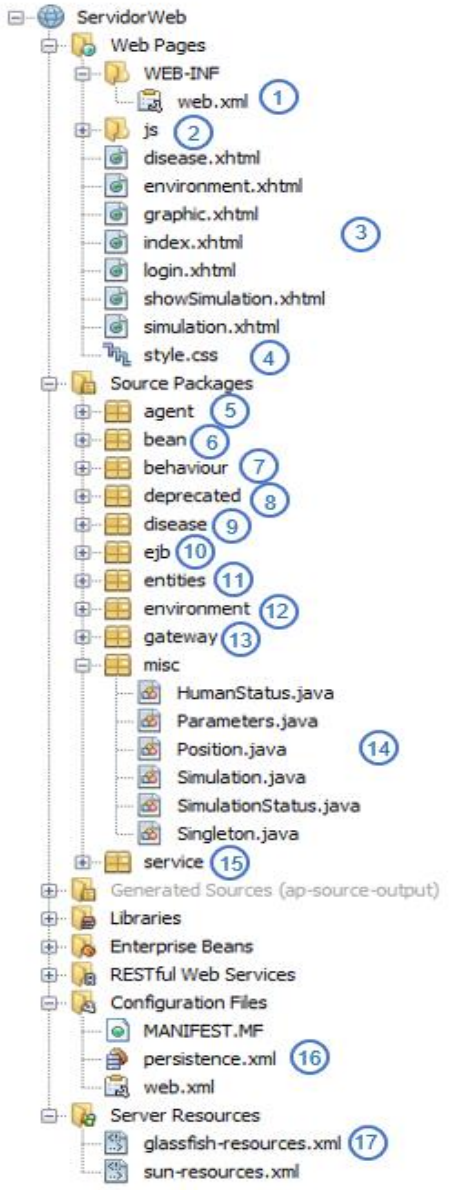

Ilustración 13 - Estructura del Proyecto de la Aplicación Servidora

1. Archivo de configuración del proyecto. En él se configuran las páginas de inicio, el tiempo de sesión y como acceder a las páginas entre otras.
2. Archivos javascript con la funciones necesarias para mostrar la simulación en el canvas.
3. Archivos xhtml. En ellos esta las distintas vistas de la página web, son los JSF Pages.
4. Archivo CSS (siglas en inglés de cascading style sheets), en el que se modifica el estilo de las distintas páginas.
5. En este paquete se encuentran todos los agentes de la simulación.
6. En el paquete bean están todos los Managed Bean que controlan las JSF Pages.
7. En behaviour se encuentra todos los comportamientos de los agentes.
8. En este paquete se encuentra el controlador antiguo que no hacía uso de los controladores secundarios, no se usa.
9. En el paquete disease se encuentra la clase que controla la enfermedad.
10. En ejb se encuentran las fachadas para acceder a la base de datos.
11. En el paquete entities se encuentra las clases que representan las entidades de la base de datos.
12. En este paquete se encuentra la clase que modela el entorno en la simulación.

13. En el paquete Gateway está el agente encargado de realizar la conexión con la plataforma JADE.
14. En este paquete hay muchas clases distintas:
    a. HumanStatus y SimulationStatus: Guardar el estado en la simulación.
    b. Parameters: Paso de parámetros entre JADE y el servidor.
    c. Position: Las distintas posiciones de las que está formada el entorno.
    d. Simulation: Clase que sirve para controlar el tiempo de la simulación.
    e. Singleton: Patrón singleton para controlar que solo haya una simulación a la vez.
15. En el paquete service se encuentran todas las fachadas que implementan los servicios RESTful.
16. Es un archivo de configuración que se le indica cual es el resource, es decir, cual es la fuente de datos.
17. Archivo autogenerado que guarda la configuración de la conexión con la base de datos. Tiene los datos necesarios para del pool de conexiones y el resource.

Ahora vamos a comentar la estructura de proyecto de la aplicación de escritorio que se puede ver en la ilustración 14.

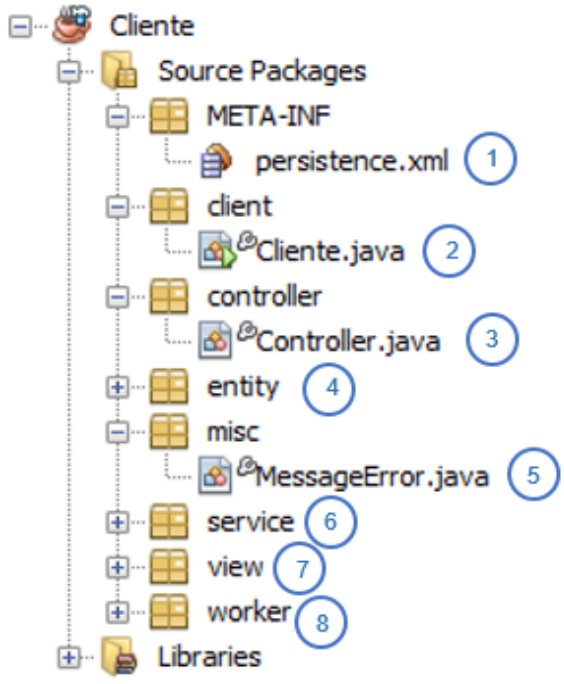

Ilustración 14 - Estructura del Proyecto de la Aplicación de Escritorio

1. Archivo de configuración en el que se indica cual es la fuente de datos.
2. Es la clase principal de la aplicación. En ella se encuentra el método main.
3. La clase Controller es la encargada de controlar las distintas vistas.
4. En el paquete entity se encuentran las clases que representan las entidades de la base de datos.
5. MessageError es una clase auxiliar utilizada para mostrar los mensajes de error.
6. En el paquete service se encuentran todas las clases que consumen los servicios RESTful.
7. En este paquete se encuentran todas las vistas de la aplicación de escritorio.
8. En el paquete worker se encuentran todas las clases que heredan de SwingWorker.

6.3. Tareas Realizadas

En este apartado se van a comentar las principales tareas desarrolladas durante la implementación de la aplicación. También hay otras muchas tareas que no hemos querido atribuírnoslas ni mencionarlas, bien porque han sido detalles insignificantes, o bien porque han sido realizadas entre todos. Además de todas estas tareas que se van a comentar a continuación, ha habido otras muchas que hemos realizado: de investigación, análisis, diseño y pruebas. En los posteriores apartados de este capítulo se van a comentar las tareas que más costosas nos han resultado a la hora de desarrollar y/o que son más interesantes.

- Tareas realizadas en la plataforma de simulación multiagentes.
    - Desarrollo de los controladores, tanto el principal, como el secundario y la comunicación entre ellos y los humanos.
    - El movimiento de los días no laborables de los humanos.
    - El movimiento libre de los humanos dentro del entorno.
    - Añadir la aceptancia a la simulación y ponerla como parámetro.
- Tareas realizadas en la aplicación servidora.
    - CRUD enfermedades.
    - Gráficas de la simulación.
    - Conexión con la plataforma Jade e inicio de simulación desde los servicios RESTful y desde la aplicación web.
    - Implementación de los Servicios RESTful de la parte del servidor.
    - Parte del desarrollo de la interfaz gráfica de usuario y css.
    - Singleton.
- Tareas realizadas en la aplicación de escritorio.
    - CRUD enfermedades.
    - Gráficas de la simulación.
    - Implementación de los servicios RESTful de la parte del cliente.
    - Desarrollo de SwingWorkers para no bloquear la hebra principal y modificación de la parte de mostrar simulación.
    - Mayor parte del desarrollo de la interfaz gráfica de usuario.

6.4. Plataforma de Simulación Multiagente

6.4.1. Controladores

Los controladores del sistema de simulación desarrollado son los encargados de enviar el número de ciclo en el que se encuentra la simulación a los agentes humanos y, al final de la simulación, recoger todos los datos de estos agentes humanos y guardarlo en la base de datos.

En un principio pensamos que lo mejor era tener un solo controlador, que cuando todos habían terminado la tarea se encargase de volver a enviarles un mensaje a todos los agentes humanos y así el sistema avanzase. Pero conforme fuimos haciendo pruebas, vimos que si lanzábamos muchos agentes, el controlador no daba abasto para tantos mensajes y el sistema se ralentizaba muchísimo, siendo poco eficiente y escalable. Se nos ocurrió que la mejor opción era hacer una jerarquía de controladores, como se puede ver en la ilustración 15.

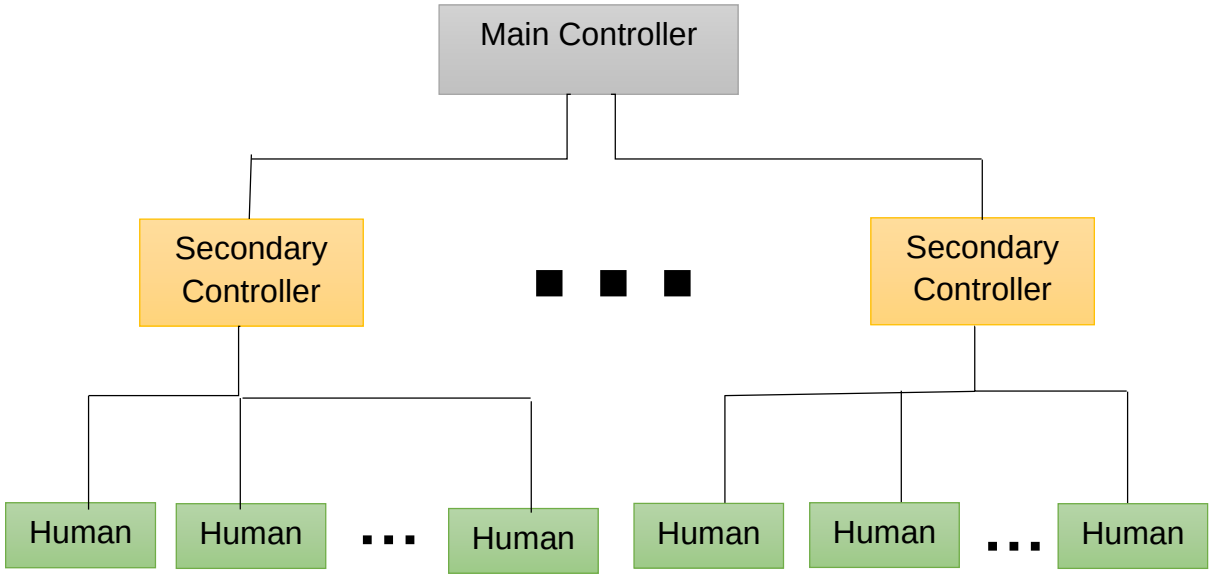

Ilustración 15 - Jerarquía de los Controladores

El controlador principal es el encargado de crear la simulación. Es el que crea todos los controladores secundarios y las instancias de la enfermedad (*disease*) y del entorno (*environment*). Al crear el entorno el controlador principal, este lanza todos los agentes humanos y los coloca en cada una de las casillas del entorno.

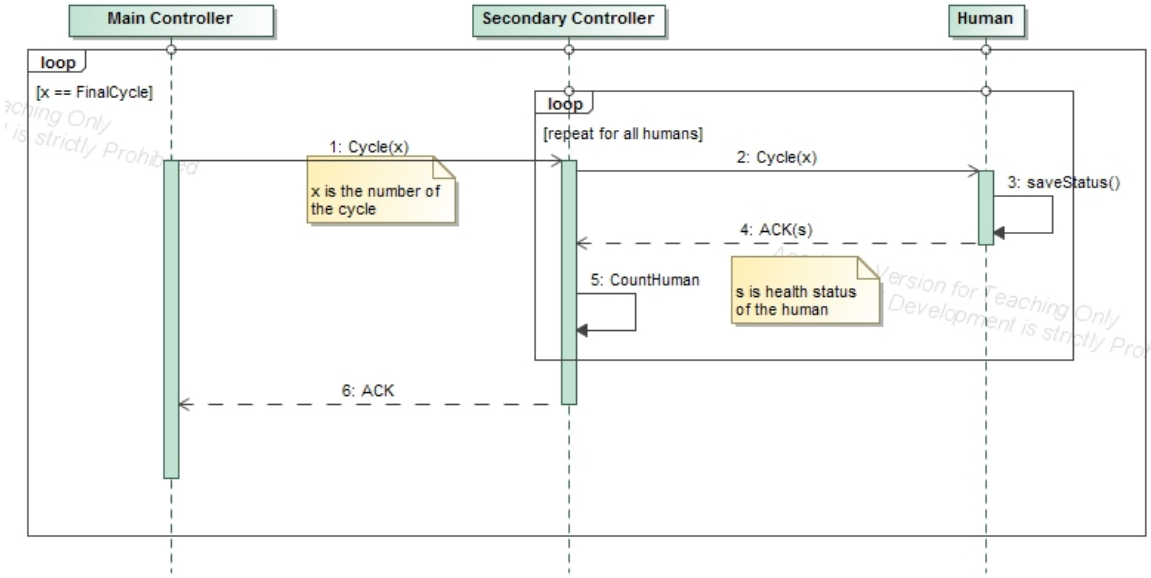

Ilustración 16 - Diagrama de Secuencia de los Controladores

Como se puede ver en la ilustración 16, lo que hace principalmente es que un controlador principal, en cada ciclo envía un mensaje a todos los controladores secundarios, y estos a su vez a los humanos que controlan. Cuando el humano lo recibe, le envía al controlador secundario un mensaje confirmando que ha recibido el ciclo y su estado de salud (*susceptible, recovered, removed, infected* o *exposed*) para recopilar datos estadísticos de la simulación.

Posteriormente, cuando un controlador secundario ha recibido la confirmación de todos los humanos, este procede a enviar la confirmación al controlador principal. Después, cuando el controlador principal ha recibido el mensaje de finalización de todos los controladores secundarios, procede a incrementar el número de ciclos y enviarle un nuevo mensaje con el ciclo a cada uno de los controladores secundarios. Este ciclo se repite hasta que termina la simulación.

Cada agente humano en cada ciclo guarda datos del estado actual del agente, es decir, su posición x e y en el tablero, su estado de salud, si está hospitalizado y su porcentaje de salud.

Cuando el controlador principal llega al último ciclo de la simulación y ha recibido la confirmación de todos los controladores secundarios, este procede a enviar un mensaje pidiendo todos los datos de la simulación a los controladores secundarios, y estos a su vez se los envían a los agentes humanos. En el momento que los agentes humanos reciben este mensaje, le envían un mensaje a su controlador secundario con todos los datos que tienen almacenados, es decir, el

estado del agente en cada ciclo. Posteriormente el agente humano finaliza junto con su hebra de ejecución.

De la misma forma, una vez que un controlador secundario ha recibido los datos de cada uno de sus agentes humanos, este los inserta en la base de datos y notifica al controlador principal. Cuando el controlador principal es notificado, les pide a los controladores secundarios los datos estadísticos que han recopilado durante la simulación y estos se los envían al controlador principal y finalizan.

Una vez que el controlador principal tiene todos los datos, este los inserta en la base de datos y finaliza.

El contenido de los mensajes tiene que ser una cadena de caracteres, por lo que los agentes antes de enviar los datos, los pasan a JSON y los envían. Esto conlleva mayor trabajo posteriormente por parte de los controladores, que se tienen que encargar de volver a convertir los datos en objetos.

6.4.2. Movimiento Día No Laborable del Humano

En la ilustración 17 se puede observar cómo ha sido la implementación del día no laborable. *A priori* parece una tarea bastante sencilla, pero su desarrollo resultó bastante más laborioso, dado que tuve que hacerlo junto con el compañero que se encargó del movimiento del día laborable, ya que compartía algunas variables y nos tuvimos que poner de acuerdo al respecto.

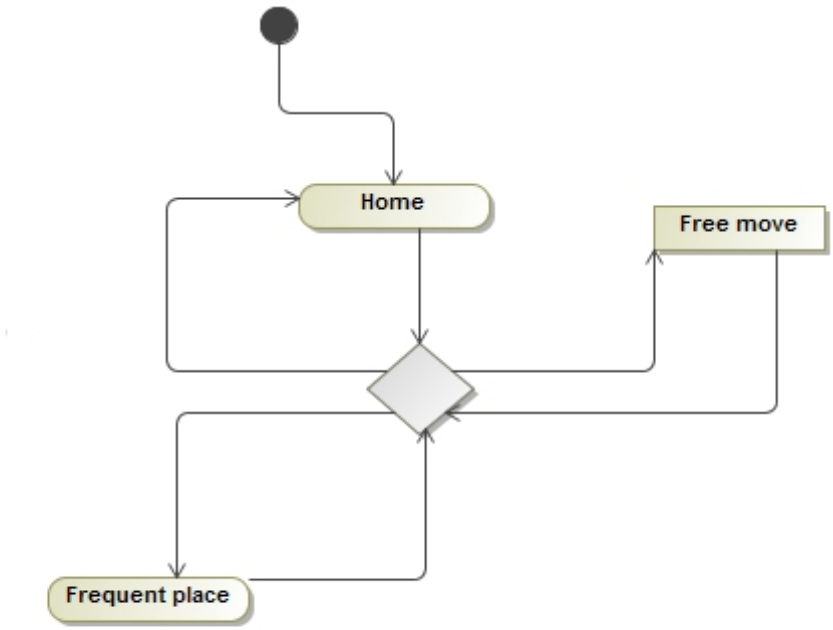

Ilustración 17 - Diagrama de Actividad Movimiento Día No Laborable

Para implementar el movimiento de día libre, hemos supuesto que el humano está durante 9 horas todos los días en su casa: 8 para dormir y 1 antes de dormir. Entonces, entre las 23:00 h. y las 8:00 h., el humano debe permanecer en su casa. Después de esa hora el humano o bien puede seguir en su casa, puede dar un paseo e ir a algún sitio aleatorio, o puede ir a un lugar frecuente. Un humano va adquiriendo más lugares frecuentes cuando ha visitado muchas veces la misma casilla.

En cada ciclo no decide si ir a otro lugar; cuando llega a un lugar saca un número aleatorio entre 1 y el número de ciclos libre que le quedan en ese día para quedarse en el mismo lugar o hacer un movimiento libre. Este movimiento de día no laborable modela también lo que hace un humano cuando ni tiene trabajo ni va a la escuela.

6.5. Aplicación Web

6.5.1. Conexión con la Plataforma JADE

Para la conexión del servidor con JADE se ha utilizado un agente Gateway. En Jade un agente Gateway tiene que extender de GatewayAgent e implementar el método *processComand*, aparte del método *setup* que lo tienen que implementar todos los agentes. El método *processComand* tiene como parámetro un objeto. Este objeto es el que se utiliza para la comunicación entre el servidor y los agentes en JADE.

En el servidor la simulación en JADE se inicia desde un Managed Bean o desde un servicio RESTful para que se pueda lanzar la simulación tanto desde la aplicación web como la aplicación de escritorio. Para arrancar la simulación, lo que se hace es lanzar este agente Gateway, que es el encargado de lanzar el controlador principal, y este a su vez inicia toda la simulación.

Para la comunicación con el agente Gateway, se ha creado la clase Parameters en el paquete Misc. Esta clase tiene todas las variables necesarias del entorno, de la enfermedad y de la simulación, con sus respectivos getter y setter, que son necesarios para poder iniciar la simulación. Esta clase Parameters también tiene una variable booleana *finished,* que es utilizada para saber si la simulación ha terminado o no.

Para asegurarnos de que en el servidor solo se está ejecutando una sola simulación, ya que esta consume la mayoría de los recursos de la máquina, se ha utilizado un patrón singleton, que está implementado en el paquete Misc cuya clase se llama **Singleton** (Ilustración 18). Como se puede ver, esta clase tiene una instancia de **Parameters**. También tiene dos funciones: una que devuelve los

```java
public class Singleton {
    private Parameters param;
    private static Singleton instance;

    private Singleton(){
        param = new Parameters();
        param.setFinished(true);
    }
    public static Singleton getInstance(){
        if(instance == null){
            instance = new Singleton();
        }
        return instance;
    }
    public Parameters getParam(){
        return param;
    }

    public boolean isFinished(){
        return param.isFinished();
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
    }
}
```

Ilustración 18 - Clase Singleton

parámetros de la simulación (*getParam*) y otra que devuelve si la simulación ha terminado (*isFinised*).

Desde el servidor para iniciar la comunicación, tanto en los servicios RESTful como en el Managed Bean, primero se comprueba si hay alguna simulación en curso. Después, si no hay ninguna simulación, se invoca a la función *getParam* de la clase Singleton y se le asignan todos los atributos adecuados a param, y la variable finished se establece a false, y así se asegura de que solo se ejecuta una simulación a la vez. Después de esto se lanza el agente Gateway con las siguientes sentencias.

```java
JadeGateway.init("gateway.AgentGateway", null);
try {
    JadeGateway.execute(params);
} catch (Exception ex) {
    ex.printStackTrace();
}
```

Ilustración 19 - Fragmento de Código de la Conexión con JADE

La primera sentencia crea al agente. En el primer parámetro, se especifica en qué paquete está y cómo se llama la clase, y en el segundo cuál es el contenedor, pero al dejarlo nulo crea un contenedor solo para este agente. La segunda sentencia sirve para iniciar la comunicación. Le pasaremos el objeto que recibirá el agente Gateway por medio del método *processComand*. Si este objeto se modifica desde la parte de JADE, es decir, de los agentes, se modificará en el servidor al usar la función *releaseComand*, que veremos posteriormente, pasándole como argumento dicho objeto de nuevo.

Como podemos observar a continuación en la ilustración 20, el método *processCommand* recibe un objeto y si es de la clase Parameters se crea un contenedor y un agente de tipo MainController, que será el encargado de lanzar toda la simulación, al que se le pasa el objeto de la clase Paramaters recibido y también el AID (identificador) del agente Gateway para después notificarle que la simulación ha terminado.

```
@Override
protected void processCommand(java.lang.Object obj) {
    if (obj instanceof Parameters) {
        System.out.println("Recibido");
        params = (Parameters) obj;
        try {

            jade.core.Runtime rt = jade.core.Runtime.instance();
            Profile p = new ProfileImpl();
            ContainerController cc = rt.createAgentContainer(p);
            Object[] arguments = new Object[2];
            arguments[0] = params;
            arguments[1] = this.getAID();
            AgentController ac = cc.createNewAgent("MainController", "agent.MainController", arguments);
            ac.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ilustración 20 - Clase AgentGateway 1

```
@Override
public void setup() {
    addBehaviour(new CyclicBehaviour(this) {
        public void action() {
            ACLMessage msg = receive();
            if (msg != null) {
                if (msg.getContent().equals("finished")) {
                    System.out.println("finished");
                    params.setFinished(true);
                    releaseCommand(params);
                    doDelete();
                }
            } else {
                block();
            }
        }
    });
    super.setup();
}
```

Ilustración 21 - Clase AgentGateway 2

En la ilustración 21 se puede ver el método *setup* (se ejecuta al iniciar el agente) del agente Gateway, le añado un comportamiento cíclico (CyclicBehaviour). Dentro de este comportamiento cíclico se hace que el agente espere bloqueado hasta recibir un mensaje. Si el mensaje recibido es “finished”, entonces es que el controlador principal ya ha terminado, lo que conlleva que la simulación ha terminado y los datos han sido almacenados en la base de datos. Entonces, pone la variable del Objeto Parameters finished a true y se lo envía al servidor con la sentencia *releaseCommand*. Seguidamente el agente Gateway muere.

6.5.2. Servicios RESTful

Para la implementación de los servicios web RESTful se ha utilizado JAX-RS, que es una API en JAVA para simplificar y facilitar el desarrollo de aplicaciones que usan arquitectura REST. En esta API se utilizan anotaciones para hacer el desarrollo más simple y fácil. Cuando los servicios RESTful son implementados, se genera automáticamente un fichero descripción WADL, que es una descripción XML de un servicio REST.

Las clases recursos son POJOs que representan las entidades de la base de datos y habían sido generadas automáticamente anteriormente a partir de la base de datos. A partir de los POJOs son generados también automáticamente los servicios REST. Los métodos autogenerados son *create, edit, remove, find, findAll, findRange* y *countRest*.

Estos métodos autogenerados son insuficientes, por lo que hay que hacer más consultas con la base de datos y exponer más métodos en el servicio.

Los servicios REST se crean a partir de clases java. Antes de declarar la clase se anota con @Path("/ruta"), que indica la ruta relativa de un URI (identificador de recursos uniformes), que identifica los recursos de una red de forma unívoca. Sobre cada método se pone también la anotación @Path("/ruta1"), que es relativa al path de la clase, es decir, este método tendrá una ruta de "/ruta/ruta1". No puede haber más de un método en un tipo de petición HTTP (GET/POST…) generando la misma representación de la respuesta y que compartan el mismo Path.

En el path también se pueden añadir variables. Se pone / y el nombre de la variable entre corchetes @Path("/ruta/{variable1}/{variable2}"). Después, en la cabecera del método se especifica con @PathParama("variable1") seguido del tipo y el nombre de la variable como se puede ver en la ilustración 22.

```
@GET
@Path("user/{name}/{idUser}")
@Produces({"application/xml", "application/json"})
public List<Simulation> findSimulationByNameAndUser(@PathParam("name")String name,@PathParam("idUser") Integer idUser) {
    return em.createQuery("SELECT s FROM Simulation s WHERE s.name = :name AND s.idUser.iduser = :idUser").
            setParameter("name", name).
            setParameter("idUser", idUser).getResultList();
}
```

Ilustración 22 - Ejemplo Query RESTful

El comportamiento de un recurso viene determinado por el método HTTP al que está respondiendo, es decir, GET, POST, PUT y DELETE, que son para leer, crear, actualizar y eliminar respectivamente. Las anotaciones sobre la cabecera son @GET, @POST, @PUT y @DELETE. Como se puede apreciar en la imagen

anterior (ilustración 22) el método *findSimulationByNameAndUser* lo que hace es leer de la base de datos(@GET) y devuelve una lista de simulaciones.

La anotación @Produces se utiliza para representar la respuesta generada para el cliente. Es posible definir un único tipo @Produces(“text/plain”) o varios tipos como se puede observar en la imagen de arriba. Los tipos básicos son “text/plain”, “application/xml”, “application/json” y también hay otros como “application/pdf, “image/png”…

Hay más anotaciones que se usan para los métodos, pero para esta aplicación solo ha sido necesario utilizar estas.

6.5.3. Graficas de la Simulación

Para las gráficas de la simulación tuve que modificar los controladores, para que recopilara los datos necesarios para dibujarlas, es decir, el número de susceptible, infectados, muertos, recuperados y en periodo de incubación en cada ciclo y que lo almacenara en la base de datos al final de esta. Es verdad que se podrían haber conseguido los datos mediante consulta a la base de datos, pero eran consultas bastante costosas y, dado que no son demasiados datos, decidimos finalmente tenerlo almacenado para hacer las consultas más rápidamente y que el tiempo empleado en dibujar las gráficas no resultara excesivo.

Para las gráficas en la aplicación web se ha empleado la librería PrimeFaces. Esta librería no ha sido utilizada únicamente para esto, sino que ha servido para otorgar un mejor aspecto visual a toda la aplicación, además de añadirle otras funcionalidades.

Para dibujar las gráficas se ha creado un objeto tipo **LineChartSeries** para cada serie de datos que hay que representar. Posteriormente se hace una consulta a la base de datos y se añaden los datos a estas series de datos y, seguidamente, se añaden estas series al modelo gráfico. También se le han añadido distintas funcionalidades, como la animación de la gráfica o la posibilidad de hacer zoom en esta, así como un botón para resetear el zoom.

6.6. Aplicación de Escritorio

6.6.1. Servicios RESTful

Los servicios RESTful del lado del cliente también son autogenerados a partir del WADL. Pero en algunos métodos generados he tenido algunos problemas y he tenido que rehacerlos. Para el desarrollo del cliente se utiliza Jersey RESTful Web Services framework, que proporciona soporte para JAX-RS.

Para el cliente, un recurso es una instancia de la Java WebTarget que encapsula una URI. Todos los métodos HTTP fijados en el servicio pueden ser invocados a partir del WebTarget y se crea de la siguiente forma.

```
public class SimulationJerseyClient {
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:10338/ServidorWeb/webresources";

    public SimulationJerseyClient() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(BASE_URI).path("entity.simulation");
    }
```

Ilustración 23 - Ejemplo Servicio RESTful Cliente 1

Como se puede observar en la ilustración 23, se crea un nuevo cliente y se le especifica cuál es su target a partir de la URI y el path concreto que se le puso a ese servicio anteriormente.

Cuando la respuesta es un solo objeto o un texto plano, los métodos autogenerados funcionan bien. En la ilustración 24 podemos apreciar que al webTarget se le indica cuál es el path del método en concreto y se le añaden las variables y, posteriormente, la función devuelve la petición que se hace, indicándole el tipo de respuesta, en este caso XML y el tipo del objeto que devuelve.

```
public <T> T findByUsername_XML(Class<T> responseType, String username) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("user/{0}", new Object[]{username}));
    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}
```

Ilustración 24 - Ejemplo Servicio RESTful Cliente 2

Pero a la hora de devolver listas no funciona bien al indicarle cuál es la clase que devuelve, por lo que hay que modificar el método e indicarle cuál es el tipo genérico concreto. Una vez hecho esto, el método ya no puede devolver un tipo genérico si no que tiene que devolver un objeto especifico, en este caso, una lista de simulaciones.

```
public List<Simulation> findByIdUser_XML(GenericType<List<Simulation>> responseType, String idUser) throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path(java.text.MessageFormat.format("user/{0}", new Object[]{idUser}));
    return resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML).get(responseType);
}
```

Ilustración 25 - Ejemplo Servicio RESTful Cliente 3

En la mayoría de los casos, para la aplicación desarrollada, las respuestas a las peticiones son listas y como hay que indicarle el tipo genérico, se ha decidido hacer métodos estáticos dentro de las clases de los servicios para no tener que hacer eso una y otra vez desde donde se vaya a utilizar. He creado un método

estático dentro de los JerseyClient para cada una de las funciones que son necesarias durante la aplicación.

Como se puede ver en la ilustración 26, lo que se hace es una instancia de la misma clase, en este caso SimulationJerseyCliente, y un GenericType de la clase que se tiene que devolver, en este caso List<Simulation>. Se llama al método que hemos hecho antes con el tipo genérico y los parámetros adecuados. Los parámetros siempre tienen que ser convertidos a String. Posteriormente se cierra la conexión y se devuelve la lista, en este caso es una lista de simulaciones asociada a un usuario. Es importante cerrar la conexión siempre.

```
public static List<Simulation> getSimulationsByUser(int idUser) {
    SimulationJerseyClient client = new SimulationJerseyClient();
    GenericType<List<Simulation>> gType = new GenericType<List<Simulation>>() {};
    List<Simulation> simulations = null;
    simulations = client.findByIdUser_XML(gType, String.valueOf(idUser));
    client.close();
    return simulations;
}
```

Ilustración 26 - Ejemplo Servicio RESTful Cliente 4

6.6.2. SwingWorker

Se han elaborado tres hebras que heredan de la clase SwingWorker, para que hagan el trabajo costoso en segundo plano y así no dejar bloqueada la hebra principal mientras esto se ejecuta. SwingWorker está presente en java desde su versión 6 y se utiliza para hacer cosas que consuman mucho tiempo. Esto se encarga de ir cambiando de hilo según resulte necesario.

Los tres SwingWorker son **WorkerDeleteSimulation**, **WorkerDoSimulation** y **WorkerShowSimulation** y sirven para borrar, hacer y mostrar una simulación respectivamente.

El más interesante es **WorkerShowSimulation,** por lo que vamos a explicarlo más detalladamente, además fue el primero que implementé para solucionar un problema que se comentará en la siguiente sección.

SwingWorker<T,V> es una clase genérica y hay que especificarle dos tipos T y V, en este caso son Boolean y List<HumanInstant>. El tipo T define el tipo que devuelve la función *doInBackground*, mientras que el tipo V define de qué tipo es el atributo del procedimiento *proccess.*

```java
public class WorkerShowSimulation extends SwingWorker<Boolean, List<HumanInstant>>{

    private int tam;
    private int idSimulation;
    private int nTick;
    private ShowSimulationView showSimulationView;
    private List<GraphicsData> listGraphicData;

    public WorkerShowSimulation(int tam,
            int idSimulation,
            int nTick,
            ShowSimulationView showSimulationView) {
        this.tam = tam;
        this.idSimulation = idSimulation;
        this.nTick = nTick;
        this.showSimulationView = showSimulationView;
    }


    @Override
    protected Boolean doInBackground() throws Exception {
        listGraphicData = GraphicsDataJerseyClient.getByIdSimulation(idSimulation);
        for (int i = 0; i < nTick; i++) {
            List<HumanInstant>  instants =
                    HumanInstantJerseyClient.getHumanInstantInATick(i, idSimulation);
            publish(instants);
            Thread.sleep(500);
        }
        return true;
    }

    @Override
    protected void process(List<List<HumanInstant>> chunks){
        List<HumanInstant> instants = chunks.get(0);
        int cycle = instants.get(0).getHumanInstantPK().getCycleNumber();
        showSimulationView.change(instants);
        showSimulationView.setData(listGraphicData.get(cycle));
    }
```

Ilustración 27 - Clase WorkerShowSimulation

La función *doInBackgrund* es lo que se ejecuta en segundo plano, en este caso lo que hace es, en primer lugar, traer la lista de los datos gráficos, que se utilizan para mostrar los datos de número de infectados, muertos, etcétera en la vista, y después un bucle for desde 0 hasta el número de ciclos que tiene la simulación. Dentro de este bucle, para cada ciclo se trae una lista con el estado de todos los humanos en ese ciclo y se le pasa como parámetro al método *publish,* justo después la hebra se queda dormida durante 500 milisegundos. El método *publish* lo que hace es ejecutar el método *process*, que normalmente se utiliza para modificar el valor de una barra de carga, pero en este caso va a actualizar la vista de la simulación.

En el método process se coge la lista de estado de los humanos, se averigua cuál es el ciclo en el que está y se actualizan los datos en **ShowSimulationView** con las funciones *change* y *setData*, que también tuve que implementarlas para poder actualizar estos datos.

Como una simulación tiene muchos datos, tanto al hacer e insertar estos datos como al eliminarlos, puede tardar bastante. Tanto **WorkerDeleteSimulation** como **WorkerDoSimulation** cuando terminan de hacer su trabajo, borrar o lanzar una simulación, muestran un mensaje y actualizan la lista de las simulaciones.

6.6.3. Graficas de la Simulación

Para las gráficas de la simulación, en la aplicación de escritorio se ha utilizado la librería JFreeChart. En este punto de la implementación ya tenía los servicios RESTful preparados para traer los datos gráficos la simulación. Lo primero que se hace es traer todos los datos gráficos asociados a esa simulación.

Después, como con PrimeFaces en la aplicación web, se crean las series de datos de manera muy similar. Todo esto se hace en el controlador (Controller). También tuve que crear una vista para mostrar las gráficas (CharView), a la que le paso las series de datos. En esta vista se crea el gráfico, se añaden las distintas series, así como distintos parámetros y opciones, y posteriormente se muestra por pantalla.

6.7. Algunos Problemas en la Implementación

- **Problemas de sincronización de las hebras de los humanos:** dichos problemas venían dados porque todos los humanos están en una posición (de la clase Position), dentro de la cual hay una lista con los identificadores de los agentes que hay en ella. Así pues, a la hora de entrar a una posición, el agente borra el identificador de esa posición y lo añade en la nueva. También se accede a esta lista cuando se quiere intentar infectar a los otros humanos de esa posición, para saber todos los que están ahí. Este error nos salía solo en algunas simulaciones, y estuvimos un tiempo buscando el fallo, hasta que nos dimos cuenta de la razón por la cual sucedía, y lo solucionamos haciendo las listas del tipo **CopyOnWriteArrayList,** que tiene protección con respecto a la lectura y escritura concurrente.
- **Conexión con JADE:** para ello, al principio había algunos conceptos que no estaban muy claros y que requirieron un proceso exhaustivo de pruebas, con el fin de verificar su comportamiento. Para esta conexión, las clases de los agentes tienen que estar en el mismo proyecto que el servidor y ya se le indica a JADE en qué paquetes están y cómo se llaman, para que entonces JADE las ejecute en su plataforma. Tuve bastantes problemas, porque en un principio el servidor iba a estar desplegado en una aplicación empresarial Java EE, para favorecer la separación entre la capa de negocio. Pero después de buscar durante mucho tiempo, no encontré ninguna información en la web, ni en inglés ni en español. Posteriormente decidimos hacerlo en una aplicación web de java, porque en ella sí conseguí hacerlo funcionar. Después de toda la información que recopilé y en todos los tutoriales que estuve mirando, solo lo hacían en un Aplicación Web, por lo que deduje que no se podía hacer en una aplicación empresarial.
- **No se mostraba la simulación en la aplicación de escritorio:** en un principio, los datos de la simulación los volcábamos en un fichero, y un compañero se encargó de hacer una pequeña aplicación para mostrar la simulación gráficamente, y funcionaba bien porque solo hacía eso. Cuando intenté integrar y adaptar lo que hizo el compañero con la aplicación de escritorio final, la simulación no funcionaba y era muy extraño. Después de mucho investigar, me di cuenta del problema: bloqueo de la hebra principal de la interfaz gráfica, por lo que solo mostraba la etapa final de la simulación.

Entonces decidí hacerlo con un SwingWorker, como si de una barra de carga se tratase, como he comentado en el capítulo anterior. De este modo, al ejecutarse todo esto en una hebra aparte, ya funcionaba perfectamente.

- **Introducir simulación en la base de datos en la aplicación web:** antes de iniciar la conexión con JADE, insertábamos la simulación en la tabla simulation, para así poder tener el id de la simulación. El problema que ocurría es que esa simulación aparecía listada, se podía visualizar en la página de simulaciones sin que hubiera terminado, por lo que no se podía ver la simulación y, al intentar hacerlo, provocaba un error. En la aplicación de escritorio no ocurría porque hice un SwingWorker que controlaba que se mostrara cuando estuviese terminada la simulación, aunque también se ha hecho para evitar problemas. La solución que encontramos fue introducir un usuario con id -1, al que le asociábamos esa simulación en un principio y, una vez terminada y estando todo almacenado en la base de datos, se cambia el usuario y entonces es cuando esta simulación le aparece al usuario.

7. Pruebas

En este proyecto no hemos podido hacer pruebas automatizadas, ya que es una aplicación muy grande y solamente el hecho de hacer este tipo de pruebas de todo el sistema podría considerarse otro proyecto diferente.

Como el proyecto se ha realizado siguiendo la metodología Scrum, después de implementar cualquier tarea hemos realizado diversas pruebas para comprobar que todo funcionara como se pretendía.

Las pruebas más interesantes que se han realizado durante el desarrollo del proyecto han sido distintas pruebas de estrés y rendimiento del sistema multiagente, para ponerle los límites a las simulaciones y comprobar lo que el sistema puede aguantar en nuestras máquinas.

Este es un sistema desarrollado para funcionar en un servidor potente, con mucha capacidad de procesamiento y de memoria. Nosotros solo lo hemos podido probar en nuestros equipos portátiles, por lo que los límites son bastante pequeños, en comparación con lo que creemos que puede soportar. Al no poder comprobarlo en este tipo de máquinas hemos tenido que limitar la aplicación para que solo funcione en los límites que nosotros hemos podido comprobar que el sistema es estable.

Las pruebas han sido realizadas en un equipo portátil con las siguientes características:

- CPU: Intel Core i7-2630QM 2.00 GHz / 2.90 GHz con Turbo boost. 4 hilos físicos y 8 hilos lógicos. 6MB de cache
- Memoria RAM: 6GB
- OS: Windows 10 Home. 64bits.

Los resultados de las pruebas más interesantes ordenadas cronológicamente se pueden ver en la tabla 20.

Número de agentes (hebras)	Número de días	Cantidad de memoria virtual (MB)	Funciona	Comentario
2000	10	512	Si	Funcionamiento muy lento.
2000	30	512	No	La simulación no llegó a completarse.
1200	30	512	Si	Pérdida de datos de 1 agente por desbordamiento de memoria.
500	60	512	Si	
750	60	512	No	Desbordamiento de memoria.
750	60	1024	Si	
1300	30	1024	Si	
2000	30	2048	Si	Mejora mucho el rendimiento con respecto a la primera prueba con 2000 agentes.
2000	60	2048	No	Desbordamiento de memoria.
2000	50	2048	Si	
1500	70	2048	Si	
1500	80	2048	No	Desbordamiento de memoria.
1000	110	2048	Si	
1000	120	2048	Si	
1000	130	2048	Si	
1000	140	2048	No	Pérdida de mensajes por desbordamiento de memoria.
500	150	2048	Si	
500	170	2048	Si	
500	200	2048	No	Pérdida de mensajes por desbordamiento de memoria.
500	185	2048	No	Pérdida de mensajes por desbordamiento de memoria.

Tabla 20 - Pruebas de Estrés

Al principio, no teníamos en cuenta la cantidad de memoria que le teníamos asignada a la máquina virtual de Java. A partir de las pruebas, como se puede observar en la tabla 20, incrementamos el valor de la memoria obteniendo resultados mejor de lo esperado. No solo aguantaba simulaciones más grandes, sino que también aumentó mucho el rendimiento.

Los resultados de las pruebas dicen que los límites del sistema son los siguientes:

- es capaz de lanzar entre 500 agentes (hebras) durante 170 días.
- es capaz de lanzar entre 500 y 1000 agentes (hebras) durante 130 días.
- es capaz de lanzar entre 1000 y 1500 agentes (hebras) durante 70 días.
- es capaz de lanzar entre 1500 y 2000 agentes (hebras) durante 50 días.

Al terminar de realizar todas las pruebas, nos hemos dado cuenta que el sistema está limitado por la cantidad de memoria RAM del equipo. Las grandes simulaciones requieren mucha memoria de la que no disponemos, por lo que hemos tenido que limitar la memoria de la máquina virtual de java a 2gb y por eso hemos tenido que limitar a su vez el número de días y de agentes.

Cuando empezamos a realizar estas pruebas de estrés, los límites eran menores, ya que en un principio solo había un controlador. Después decidimos añadir la jerarquía de controladores, con lo que mejoramos mucho el rendimiento del sistema.

Posteriormente hicimos otra mejora, ya que al ser el controlador principal el que insertaba todos los datos en la base de datos, por lo que tenía que recibir todos los datos y convertirlos a objetos, hacía que se desbordara la memoria virtual. Conseguimos mejorar mucho los resultados haciendo que fuesen los controladores secundarios los que insertaran los datos de los de los humanos que controlaba.

También estuvimos haciendo distintas pruebas de rendimiento en las inserciones en la base de datos, ya que al principio, una simulación de 3 días y 300 agentes tardaba 17 segundos aproximadamente. Lo que nos pareció que era una cantidad de tiempo bastante grande. Conseguimos solucionarlo borrando la clave foránea que apuntaba a desde la tabla HumanInstant hacia Simulation. Haciendo ese simple cambio, paso de tardar 17 segundos a tardar menos de medio segundo.

8. Conclusiones y Mejoras Futuras

8.1. Conclusiones

Este proyecto en el que nos hemos embarcado mis dos compañeros y yo, ha sido un duro reto, ya que era un proyecto multidisciplinar que abarca desde hacer un servidor web, grandes bases de datos, programación orientada a agentes, conocimiento de las técnicas de simulación y de los modelos de epidemia, servicios web, etc.

Uno de los grandes problemas al principio es que había mucha información que desconocíamos, por lo que antes de empezar el proyecto en sí, hemos tenido que hacer un gran trabajo de investigación, tanto para elegir las herramientas y tecnologías necesarias, como para aprender a cómo usarlas posteriormente y adquirir muchos conceptos con los que no estábamos familiarizados.

La programación orientada a Agentes ha sido todo un reto para nosotros, ya que es una nueva forma de programar a la que no estábamos habituados. Al principio, veíamos a los agentes como objetos, lo que nos llevó a equivocarnos en más de una ocasión en el diseño. Pero una vez que interiorizamos bien el concepto, fuimos avanzando mucho más rápidamente.

Por cada simulación se necesitan muchísimas entradas en la base de datos, por lo que se ha tenido que optimizar todo lo posible tanto el modelo como la configuración de la misma, lo que ha requerido también un gran esfuerzo extra.

También hemos apreciado que un proyecto de esta envergadura requiere mucha planificación y sincronización por parte de todos, para que las cosas salgan bien; hace falta que todo esté bien organizado y planteado desde el principio porque un pequeño fallo al inicio puede hacer que haya que modificar muchas cosas en un futuro, como nos ha ocurrido, pero espero que esto nos sirva como experiencia.

Para concluir, ha sido un proyecto muy costoso para todos, pero a la vez me ha enseñado muchas cosas que quizás, de no haber sido por haberlo realizado, no hubiera aprendido nunca, y me ha aportado mucha experiencia sobre el trabajo en grupo y la toma de decisiones. Si pudiera volver a elegir un nuevo proyecto, volvería a elegir este porque, como he comentado antes, aunque no haya sido un proyecto fácil de realizar me ha aportado muchas cosas y lo mejor de todo es la gratificación personal cuando he comprobado lo que hemos sido capaces de realizar.

8.2. Mejoras Futuras

Durante el desarrollo, se nos han ocurrido diferentes ideas sobre qué añadirle al proyecto, pero por el coste temporal del rediseño y desarrollo, las hemos descartado ya que se nos sale del límite del proyecto. Algunas de ellas son:

- **Añadir transporte público al sistema:** ahora mismo los agentes humanos no tienen la posibilidad de contagiarse mientras viaja de un lugar a otro lejano, es decir, para este proyecto hemos supuesto que si tiene que viajar muy lejos cada agente tiene su vehículo personal y va de un lugar a otro sin contagiarse.
- **Añadir grandes zonas de trabajo y colegios:** habilitar zonas de trabajo que sean mucho más grandes que quizás ocupen varias casillas (como una especie de casilla más grande) donde la gente va a trabajar o al colegio y está unas horas concretas moviéndose por esas zonas. Esto quizás aumentaría el contagio haciendo la simulación más real.
- **Introducir más seres vivos:** introducir por ejemplo animales que puedan portar la enfermedad y que puedan contagiar a los humanos, sería una buena posible mejora. Pensando en esto hicimos la clase LivingBeing (ser vivo) para que en un futuro se pudieran incorporar los animales a las simulaciones.
- **Incluir grupos de usuarios**: hacer que distintos usuarios pertenezcan a un grupo de usuarios que puedan compartir simulaciones, entornos y enfermedades.
- **Modelizar los datos**: a partir de una serie de datos estadísticos modelizarlos de manera que las fórmulas para contagiar, calcular número de días de infección o de exposición y la probabilidad de morir, se mejoren todo lo posible.
- **Mejorar movimiento de los humanos**: actualmente, los humanos se mueven de un lugar a otro directamente, menos si se mueven libremente por el entorno. Se podría implementar el movimiento del humano siga una heurística determinada.
- **Simulación con varias enfermedades**: hacer simulaciones en las que los humanos se puedan contagiar con distintas enfermedades a la vez.
- **Simulación de más de un entorno**: JADE es una plataforma que permite distribuir los agentes en distintos equipos. Sería muy interesante poder ejecutar distintos entornos en distintas máquinas y que los agentes se pudieran mover de un entorno a otro. Esto haría que las simulaciones fueran mucho más realistas.
- **Hospitalización de enfermos**: un humano enfermo puede ser hospitalizado. Esto permitiría disminuir el número de días infectados, la probabilidad de muerte y de contagio a otros humanos.

- **Vacunas:** la posibilidad de introducir un numero inicial de personas vacunadas. Esto haría que un grupo de personas fuese inmune a la enfermedad desde el principio.

Referencias Bibliográficas

The Journal of Artificial Societies and Social Simulation), An Agent-Based Spatially Explicit Epidemiological Model in MASON, realizado por Jill Bigley (2005) http://jasss.soc.surrey.ac.uk/9/1/3.html

Simulación de procesos sociales basada en agentes software EMPIRIA. Revista de Metodología de Ciencias Sociales. N. o 14, julio-diciembre, 2007, pp. 139-161. ISSN: 1139-5737 http://samer.hassan.name/files/Empiria.pdf

Historia de la simulación: http://www.landersimulation.com/formacion-con-simulacion/el-mundo-en-movimiento/historia-de-la-simulacion/

Wikipedia: https://es.wikipedia.org/

Tutoriales y guías de JADE: http://jade.tilab.com/documentation/tutorials-guides/

Guía de programación en JADE: https://programacionjade.wikispaces.com/

StackOverflow: http://stackoverflow.com/

Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood (2007): Developing Multi-Agent Systems with JADE. WILEY

Ana Mas. (2005). Agentes Software y Sistemas Multiagente: Conceptos, Arquitecturas y Aplicaciones

Java API: http://docs.oracle.com/javase/7/docs/api/

JADE API: http://jade.tilab.com/doc/api/

PrimeFaces API: http://www.primefaces.org/docs/api/5.2/

JFreeChart API: http://www.jfree.org/jfreechart/api/javadoc/index.html

JSF API: https://javaserverfaces.java.net/docs/2.2/javadocs/

JAX-RS API: https://jax-rs-spec.java.net/nonav/2.0-rev-a/apidocs/index.html

Jersey API: https://jersey.java.net/apidocs/latest/jersey/

GSON API: https://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/com/google/gson/Gson.html

MASON: https://cs.gmu.edu/~eclab/projects/mason/

REPAST: http://repast.sourceforge.net/

Apéndice

A) Manual de Instalación

Requisitos Mínimos

Para poder ejecutar el sistema la máquina virtual de Java debe tener al menos 2GB de RAM asignados.

Java

En el caso del proyecto desarrollado, la versión mínima soportada para Java es 1.7, por lo que será necesario descargarla desde la web e instalarla en nuestro sistema si no disponemos de ella previamente o disponemos de una versión inferior.

Para descargar la última versión de Java podemos hacerlo directamente en la siguiente URL: https://www.java.com/es/download/ , y seguir las instrucciones del proveedor dependiendo de la plataforma en la que se desee trabajar.

Base de Datos

La base de datos utilizada en el proyecto es SQL por lo que podría usarse cualquier solución que soporte este tipo de base de datos. En nuestro caso, recomendamos MySQL de Oracle, ya que puede obtenerse de forma gratuita y aporta todas las funcionalidades que requiere el proyecto además de ser fácilmente configurable por lo que se puede optimizar el servidor para que su comportamiento sea el más adecuado para el uso que se hace del desde el sistema desarrollado.

El servidor de bases de datos puede descargarse desde http://dev.mysql.com/downloads/mysql/. El servidor está disponible para distintos sistemas operativos como por ejemplo Linux, Mac o Windows.

Durante la instalación se solicita el puerto en el que se desea que trabaje el servidor, en este caso, se recomienda usar el que MySQL usa por defecto (3306). También se solicita una contraseña para el usuario administrador ‘root’ que tiene que ser vacía, aunque se podría asignar alguna configurando la aplicación.

Las aplicaciones desarrolladas, tanto el cliente java como la aplicación servidora se conectan usando el usuario ‘root’, pero podríamos crear un usuario expresamente para esta, siempre que le asignemos permisos de lectura, modificación y borrado.

JADE

JADE en una framework Java que actúa como gestora del ecosistema en el que los agentes actúan, que se encarga tanto de albergar a los agentes en contenedores y plataformas como de gestionar la comunicación entre ellos.

Para que la aplicación servidora desarrollada pueda funcionar es necesario tener previamente en ejecución la aplicación Java de JADE y, además haber importado JADE como librería Java importando al proyecto los dos ficheros precompilados con extensión .jar que pueden ser descargados desde http://jade.tilab.com/download/jade/license/jade-download/?x=34&y=1 descargando el paquete 'jadeBin'.

En cualquier caso, a la hora de importar el proyecto, la IDE (entorno de desarrollo) usada identificará la dependencia existente con esta librería y solicitará que se indique la localización de la misma para poder construir el proyecto.

- **Incorporación al proyecto de la librería JADE**

Una vez descargados los ficheros .jar, pueden ser incorporados al proyecto como se haría normalmente en el IDE utilizado.

En el caso de NetBeans, será necesario hacer clic derecho sobre el proyecto (Server) y seleccionar la opción "Propiedades". Esto nos abrirá la siguiente ventana.

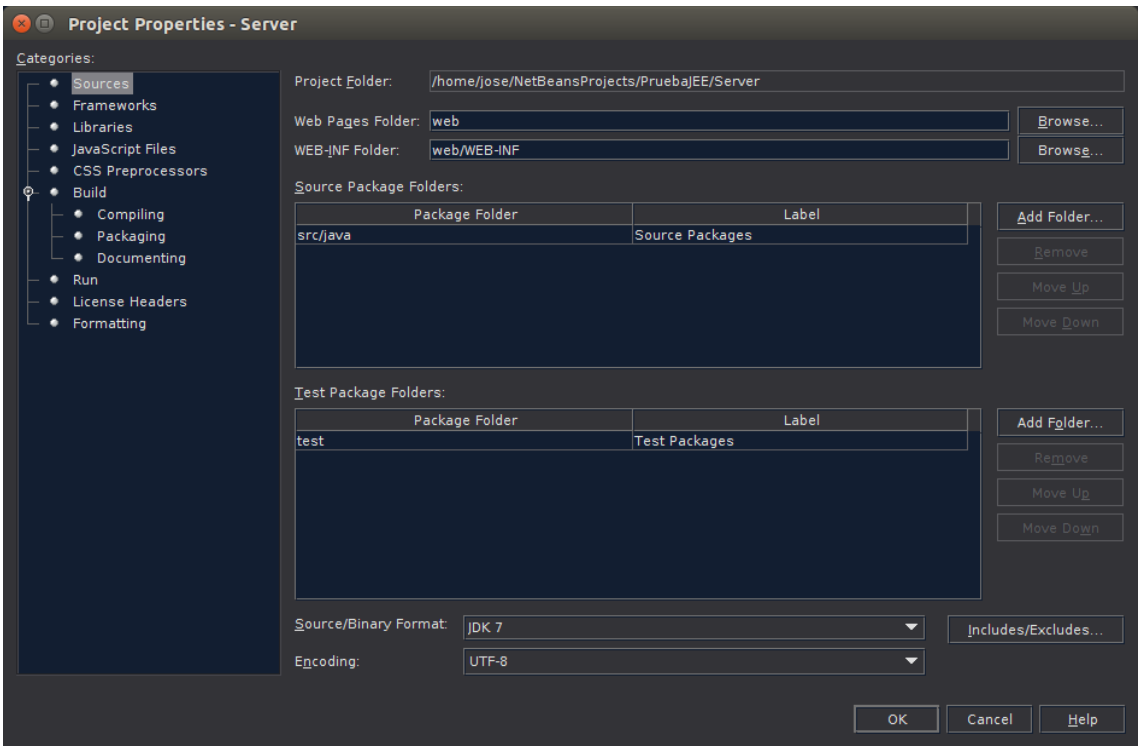

Ilustración 28 - Propiedades del Proyecto

Si hacemos clic sobre la opción “Librerías” en el panel izquierdo obtendremos la siguiente vista, donde si pulsamos en el botón “Add JAR/Folder” nos permitirá añadir al proyecto las librerías que necesitemos en forma de ficheros .jar.

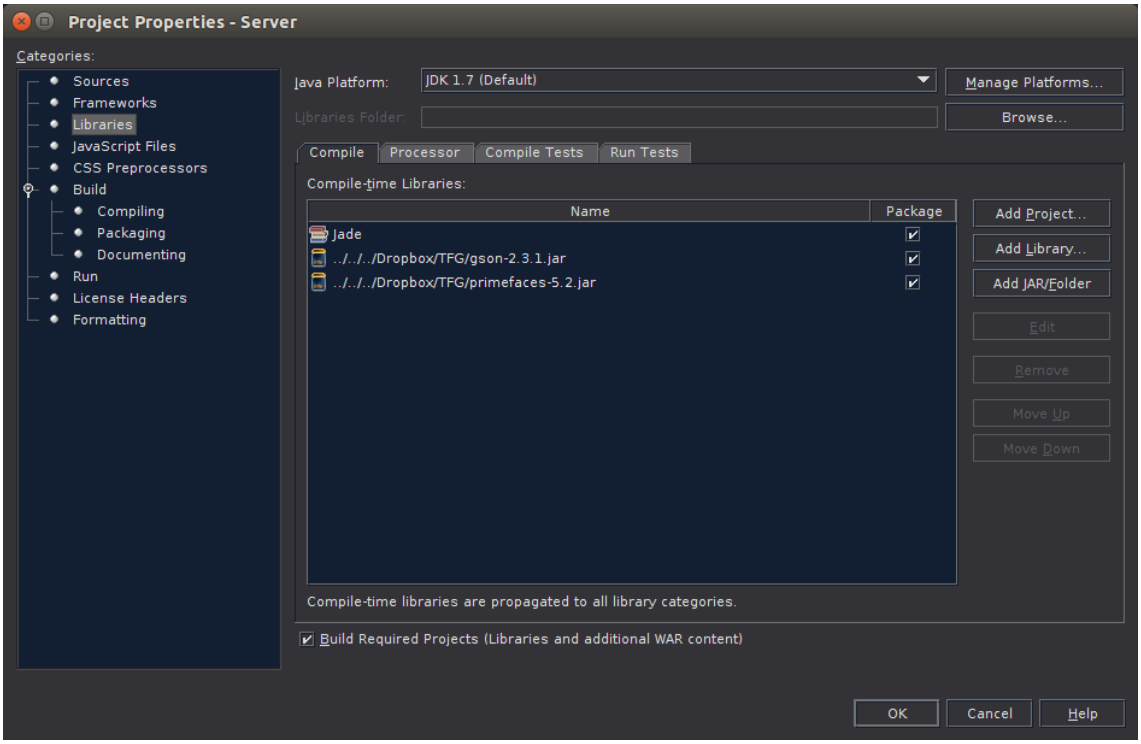

Ilustración 29 - Librerías que Contiene el Proyecto

- **Ejecución de la aplicación Jade**

Para la ejecución de la aplicación Jade, es necesario hacer uso de la interfaz de consola de java y una vez posicionados en el directorio en el que se encuentre el fichero jade.jar ejecutar el siguiente comando (independientemente de la plataforma en la que nos encontremos).

Para arrancar JADE hay que seguir los siguientes pasos:

1. Abrir Terminal o Símbolo de sistema.
2. Dirigirse al directorio donde se encuentra la librería de JADE (fichero jade.jar).
3. Ejecutar el siguiente comando:

```
java -cp jade.jar jade.Boot -gui
```

Esto nos abrirá la siguiente ventana donde podemos observar el estado de todo el sistema multiagente, ya sea tanto las plataformas, contenedores y agentes creados como el paso de mensajes que se producen entre estos entre otra información.

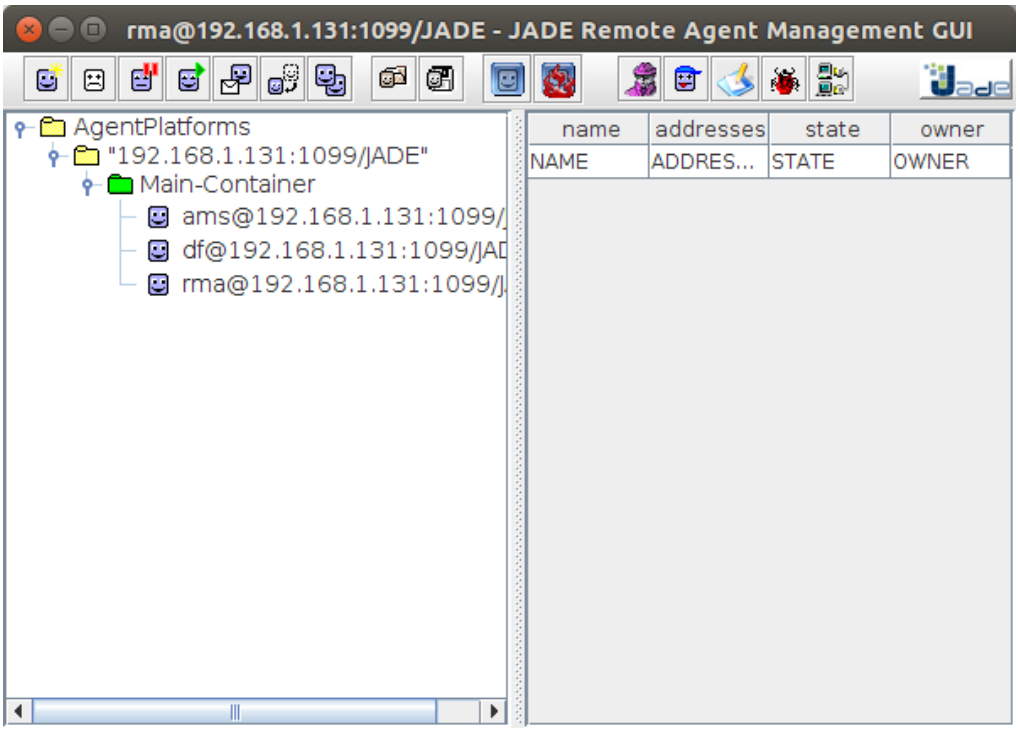

Ilustración 30 - Interfaz Gráfica de JADE

Es importante resaltar de nuevo que es necesario tener en ejecución esta aplicación si se desea que la aplicación servidora desarrollada pueda realizar nuevas simulaciones.

GlassFish

Ya que la aplicación servidora se ha desarrollado usando la tecnología JavaEE, es necesario usar un servidor que la soporte. Actualmente existen diversas opciones gratuitas como por ejemplo TomEE o GlassFish, aunque en nuestro caso, hemos decidido usar la último puesto que además de haber sido más fácil de configurar y ocasionarnos menos problemas a la hora de desplegar el proyecto en él, es posible descargar una versión de NetBeans (que es el IDE usado para el desarrollo) con este servidor preinstalado.

- **Configuración de la máquina virtual de java en GlassFish**

Es necesario asignarle como mínimo 2GB de memoria a la máquina virtual de java en GlassFish para que la aplicación funcione correctamente. Por defecto tiene asignado 512MB. Para cambiar el valor abrimos la consola de GlassFish y nos vamos a “Configurations -> JVM Settings -> JVM Options”. Buscamos el valor “-Xmx512” que es el que viene por defecto y ponemos “-Xmx2048”.

Posteriormente le damos a “Add JVM Option” y añadimos “-XX:-UseGCOverheadLimit” y pulsamos el botón Save para guardar la configuración. Con este parámetro se desactiva el error “OutOfMemoryError” que se produce

cuando más del 98 % del tiempo de ejecución se gasta en la recolección de basura y se recupera menos del 2% del montículo (*Heap)*.

Incorporación de Otras Dependencias al Proyecto

Además de las librerías de JADE que se ha comentado previamente, es necesario añadir otras que el proyecto necesita para su funcionamiento. Para ello, siguiendo el mismo proceso que para JADE, debemos añadir las librerías 'Gson' y 'Primefaces', que pueden ser descargadas en las siguientes URLs respectivamente:

http://www.primefaces.org/downloads

http://search.maven.org/#artifactdetails|com.google.code.gson|gson|2.3.1|jar

Durante el desarrollo del proyecto, la versión de 'Primefaces' usada es 5.2.10 y en el caso de 'Gson' 2.3.1.

En el caso de la aplicación cliente, también será necesario descargar las librerías 'Jcommon' y 'Freechart'. En el caso de 'Jcommon' se ha usado la versión 1.0.23 y en el de 'Freechart' 1.0.19 que pueden ser descargadas en las siguientes URLs:

http://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/1.0.19/

http://sourceforge.net/projects/jfreechart/files/3.%20JCommon/1.0.23/

B) Manual de Usuario Aplicación Escritorio

Login

Inicialmente la ventana de Login, como se puede ver en la ilustración 31.

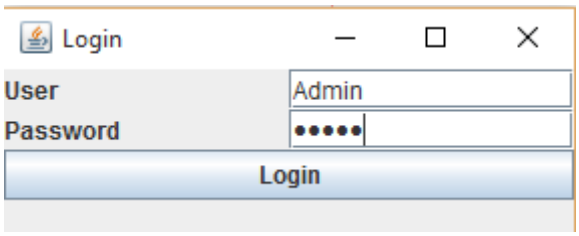

Ilustración 31 - Ventana Login

Se introduce el nombre de usuario y la contraseña y se pulsa el botón “Login” para acceder a la ventana principal de la aplicación. Si el nombre de usuario y la contraseña no son correctos se mostrará el mensaje que se puede ver en la ilustración 32.

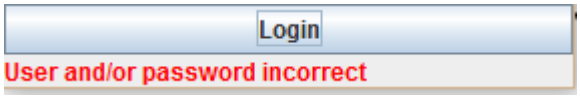

Ilustración 32 - Mensaje Login Incorrecto

Ventana Principal

En la ventana principal es la que se puede visualizar, crear, editar y eliminar tanto entornos como enfermedades. A la vez que se pueden hacer simulaciones y ver los datos de esta. La ventana principal de la aplicación se puede ver en la ilustración 33.

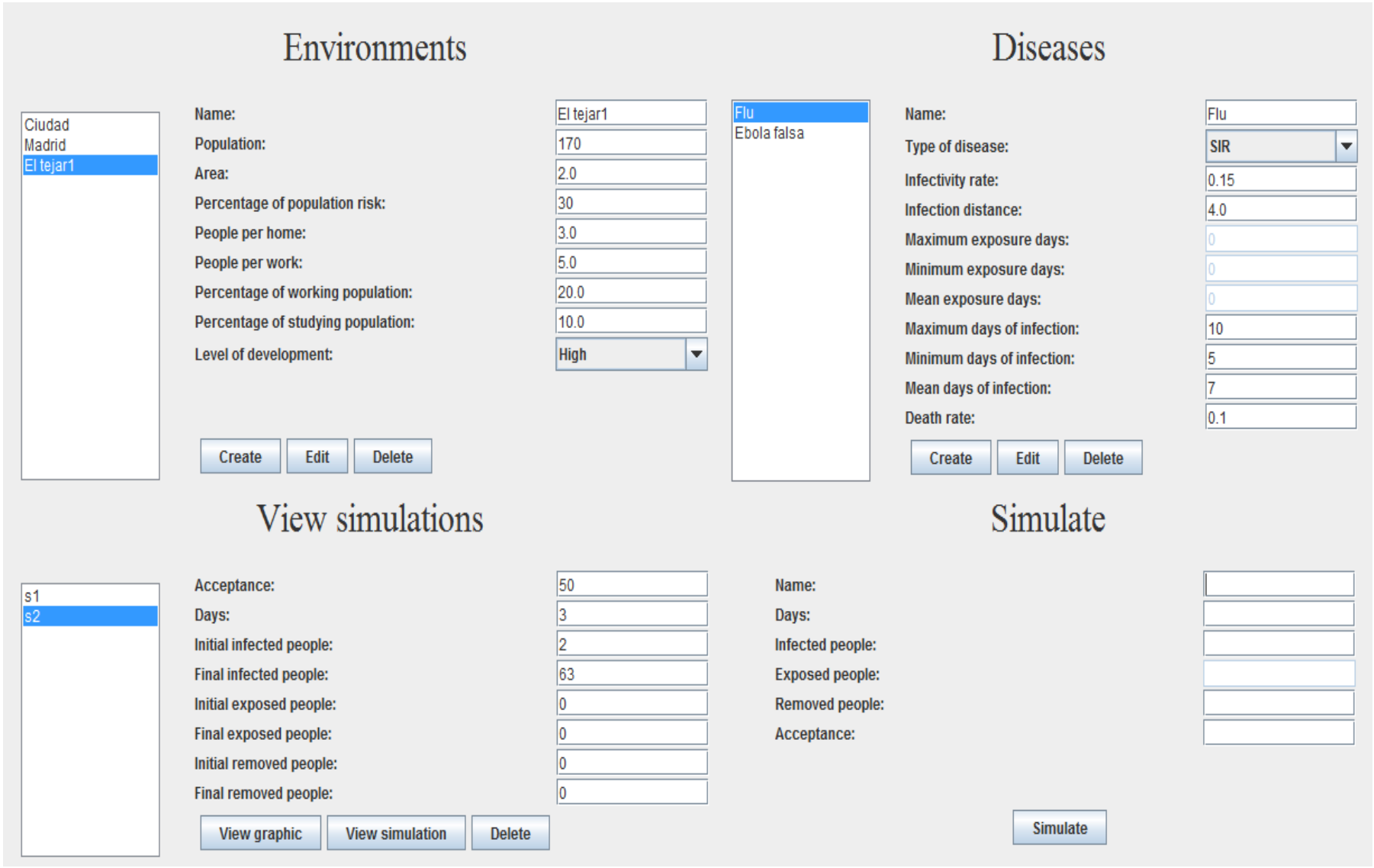

Ilustración 33 - Ventana Principal

Visualizar, Crear, Editar y Eliminar un Entorno

La parte de la ventana donde se pueden crear, editar y eliminar un entorno, y a su vez visualizar los datos de este es la que se puede ver en la ilustración 34.

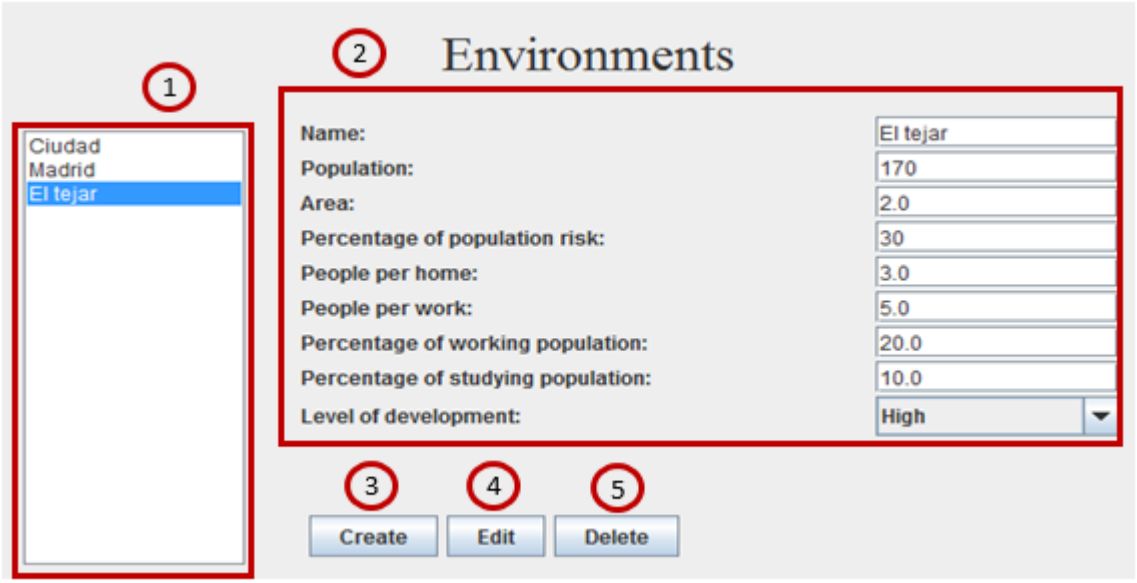

Ilustración 34 - Vista de Entornos

1. Lista de entornos: Aquí se muestra un listado con todos los entornos de un usuario. Al hacer doble clic en cualquiera de los entornos de la lista se mostrarán los datos correspondientes al mismo.
2. Datos del entorno: En este formulario se muestran todos los datos de un entorno al seleccionarlo. También sirve para introducir los datos de un nuevo entorno y para editar el seleccionado. Tanto para editar como para crear todos los campos deben estar rellenos con valores correctos.
   - Name: Nombre del entorno. Puede contener entre 1 y 15 caracteres. Un usuario no puede tener dos entornos con el mismo nombre.
   - Population: Número de personas que habita en el entorno. Tiene que ser un número entero entre 1 y 2000.
   - Area: Es el área del entorno en km^2. Es un número entre 0 y 25.
   - Percentage of population risk: porcentaje de personas de riesgo. Es un número entero entre 0 y 100.
   - People per home: Media de personas por casa. Es un número real entre 1 y el número de personas del entorno.
   - People per work: Media de personas por trabajo. Es un número real entre 1 y el número de personas del entorno.
   - Percentage of working population: Porcentaje de personas trabajando. Es un número real entre 0 y 100.

- Percentage of studying population: Porcentaje de personas estudiando. Es un número real entre 0 y 100. La suma de este campo y del anterior no puede sumar más de 100.
- Level of development: Es el nivel de desarrollo del entorno. Se puede seleccionar entre Low (bajo), medium (medio), High (alto) y Very high (muy alto).

3. Crear Entorno (Create): Este botón sirve para crear un nuevo entorno con los datos que hay en el formulario de entornos.
4. Editar entornos (Edit): Este botón sirve para editar el entorno seleccionado con los datos que hay en el formulario de entornos.
5. Eliminar entornos (Delete): Este botón sirve para eliminar un entorno seleccionado.

Tanto para crear como para editar se comprueba que todos los datos sean correctos y si no son correctos se mostrará su correspondiente mensaje de error, como se puede ver en la ilustración 35.

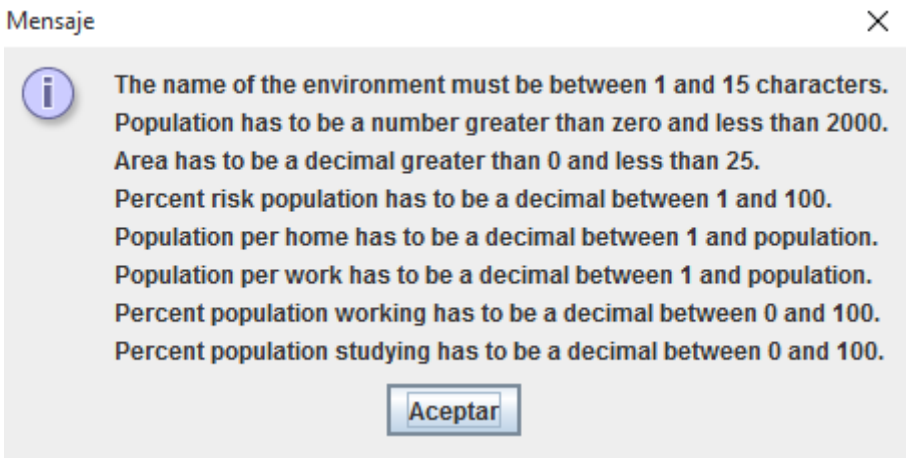

Ilustración 35 - Error de Vista de Entornos

Visualizar, Crear, Editar y Eliminar una Enfermedad

La parte de la ventana donde se pueden crear, editar y eliminar un entorno, y a su vez visualizar los datos de esta es la que se puede ver en la ilustración 36.

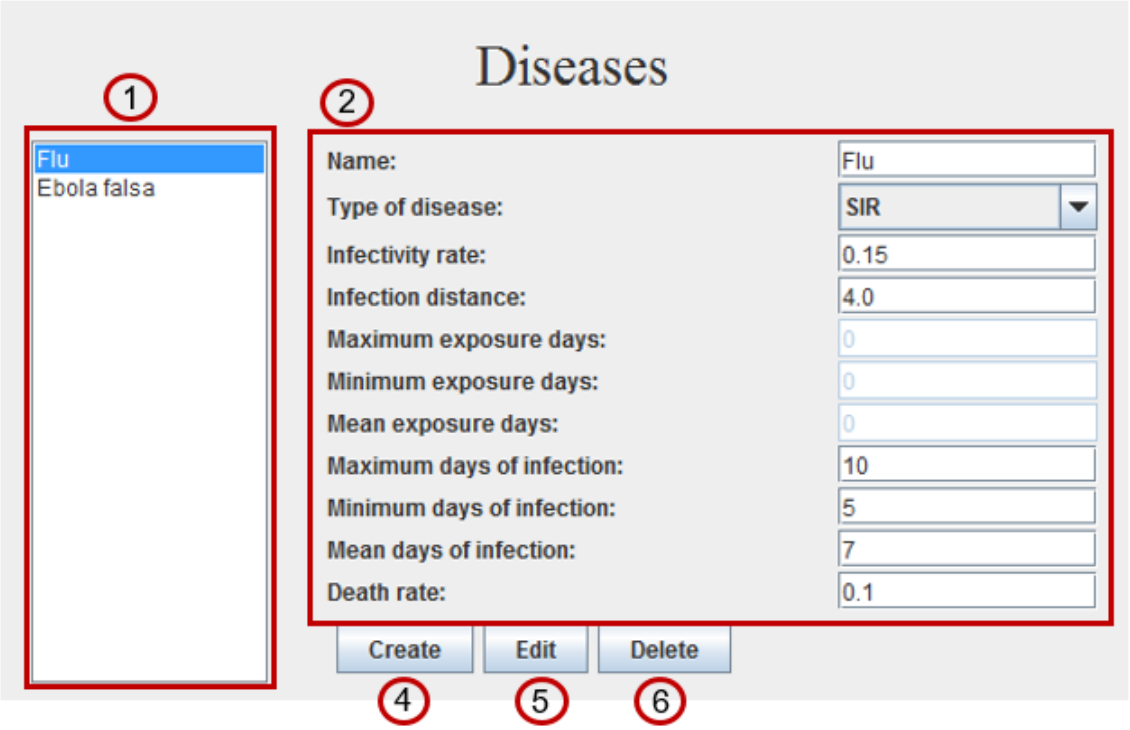

Ilustración 36 - Vista de Enfermedades

1. Lista de enfermedades: Aquí se muestra un listado con todas las enfermedades de un usuario. Al hacer doble clic en cualquiera de ellas se mostrarán los datos correspondientes a esa enfermedad.
2. Datos de la enfermedad: En este formulario se muestran todos los datos de una enfermedad al seleccionarlo. También sirve para introducir los datos de un nuevo entorno y para editar el seleccionado. Tanto para editar como para crear todos los campos deben estar rellenos con valores correctos.
   Si la enfermedad es de tipo SEIR o SEIS, todos los campos son obligatorios, en cambio, si la enfermedad es de tipo SIR o SIS son todos obligatorios excepto Maximun, Minimun y Mean exposure days.
   - Name: Nombre de la enfermedad. Puede contener entre 1 y 15 caracteres. Un usuario no puede tener dos enfermedades con el mismo nombre.
   - Type of disease: Es el tipo de enfermedad. Se puede seleccionar entre SIR, SIS, SEIR y SEIR.
   - Infection distance: Es la distancia de infección. Es un número entero entre 1 y 100.
   - Maximun exposure days: Es el periodo máximo de incubación de la enfermedad. Es un número mayor que 0.

- Minimun exposure days: Es el periodo mínimo de incubación de la enfermedad. Es un número mayor que 0.
- Mean exposure days: Es el periodo medio de incubación de la enfermedad. Es un número mayor que 0.
  Maximun exposure days tiene que ser mayor o igual que *Mean exposure days* y este a su vez mayor o igual que *Minimun exposure days*.
- Maximun days of infection: Es el periodo máximo de infección. Es un número mayor que 0.
- Minimun days of infection: Es el periodo mínimo de infección. Es un número mayor que 0.
- Mean days of infection: Es el periodo medio de infección. Es un número mayor que 0.
  Maximun days of infection tiene que ser mayor o igual que Mean days of infection y este a su vez mayor o igual que *Minimun days of infection*.
- Death rate: es el ratio de muerte. Es un número entre 0 y 1.

3. Crear enfermedad (Create): Este botón sirve para crear una nueva enfermedad con los datos que hay en el formulario de entornos.
4. Editar enfermedad (Edit): Este botón sirve para editar la enfermedad seleccionada con los datos que hay en el formulario de entornos.
5. Eliminar enfermedad (Delete): Este botón sirve para eliminar una enfermedad seleccionada.

Tanto para crear como para editar se comprueba que todos los datos sean correctos y si no son correctos se mostrará su correspondiente mensaje de error, como se puede ver en la ilustración 37.

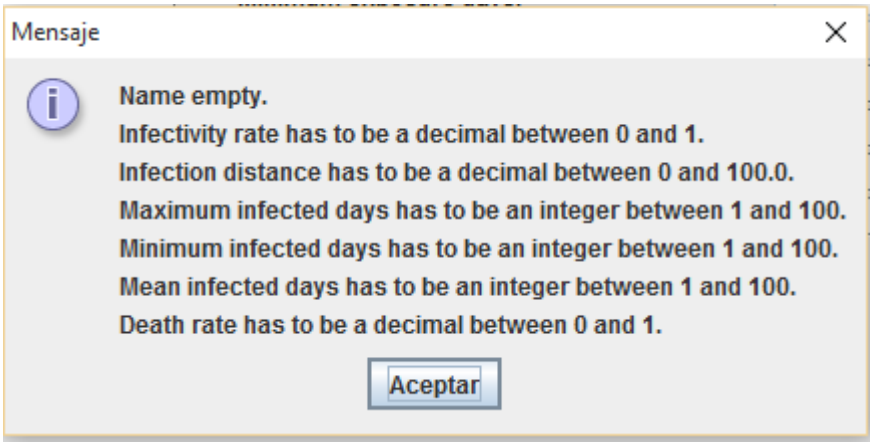

Ilustración 37 - Error de Vista de Enfermedades

Simular

La parte de la ventana donde hacer una simulación es la que se puede ver en la ilustración 38.

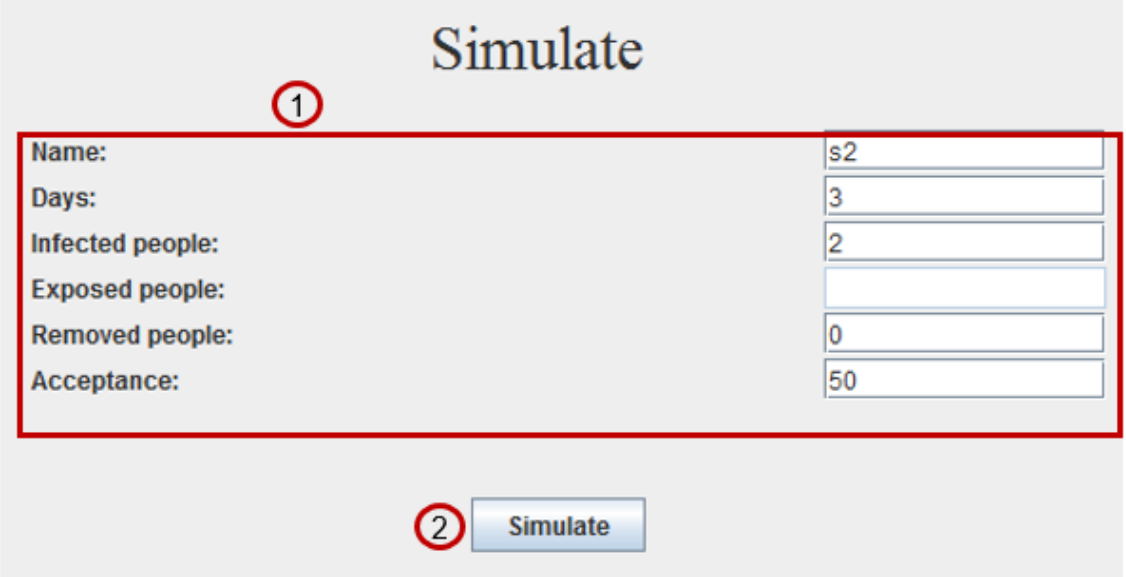

Ilustración 38 - Vista para Hacer Simulación

Primero: Se debe seleccionar una enfermedad en la lista de enfermedades y un entorno en la lista de entornos, de los dos fragmentos de la ventana comentados anteriormente.

Segundo: Se debe rellenar el formulario y pulsar el botón Simulate como se explica a continuación.

1. Formulario para hacer simulaciones. Se deben introducir todos los campos con valores correctos para poder realizar la simulación. Si la enfermedad seleccionada es de tipo SEIR o SEIS, todos los campos son obligatorios, en cambio, si la enfermedad es de tipo SIR o SIS son todos obligatorios excepto Exposed people.
   - Name: Nombre de la simulación. Puede contener entre 1 y 15 caracteres. Un usuario no puede tener dos simulaciones con el mismo nombre.
   - Days: Número de días de la simulación. El número mínimo de días es 1, y el máximo se establece en función del número de habitantes del entorno seleccionado.
     - Entre 0 y 500 humanos. 170 días.
     - Entre 500 y 1000 humanos. 130 días.
     - Entre 1000 y 1500 humanos. 70 días.
     - Entre 1500 y 2000 humanos. 50 días.

- Infected people: Número de personas infectadas. Es un número mayor que 0.
- Exposed people: Número de personas incubando la enfermedad. Es un número mayor que 0.
- Removed people: Número de personas muertas. Es un número mayor que 0.
  La suma del número de personas muertas, infectadas e incubando la enfermedad no puede ser mayor que el número de habitantes del entorno seleccionado.
- Acceptance: La probabilidad de que una persona no vaya a trabajar por que considere que está enfermo. Es un número entero entre 1 y 100.

2. Hacer simulación (Simulate). Este botón sirve para comenzar una simulación. Al pulsar el botón se comprueba que todos los datos introducidos son correctos y si no son correctos se muestra un mensaje como se puede ver en la ilustración 39.

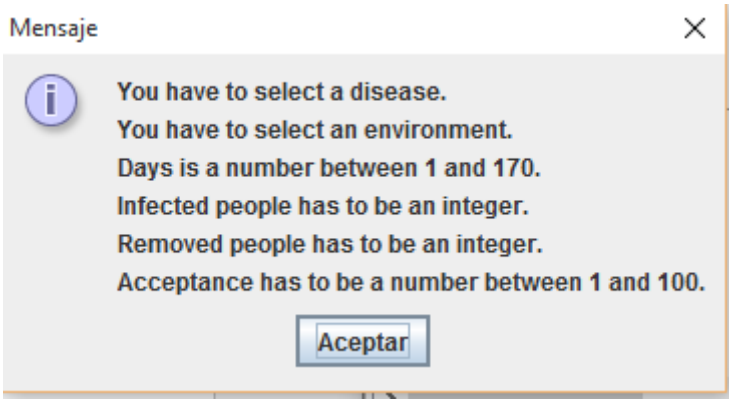

Ilustración 39 - Mensaje de Error Vista Hacer Simulación

Ver Datos de Simulación, Borrar Simulación, Ver grafica de Simulación y Ver Simulación

La parte de la ventana donde se pueden ver los datos de una simulación es la que se puede ver en la ilustración 40.

Ilustración 40 - Vista de las Simulaciones

1. Lista de entornos: Aquí se muestra un listado con todas las simulaciones de un usuario. Al hacer doble clic en cualquiera de ellas se mostrarán los datos correspondientes a esa simulación. También se autoseleccionarán la enfermedad y entorno correspondiente.
2. Datos de la simulación: En este formulario se muestran todos los datos de una simulación al seleccionarla.
3. Ver grafica (View graphic): Este botón sirve para ver una gráfica de la simulación. Al pinchar se abre en una ventana aparte como se puede ver en la ilustración 41.

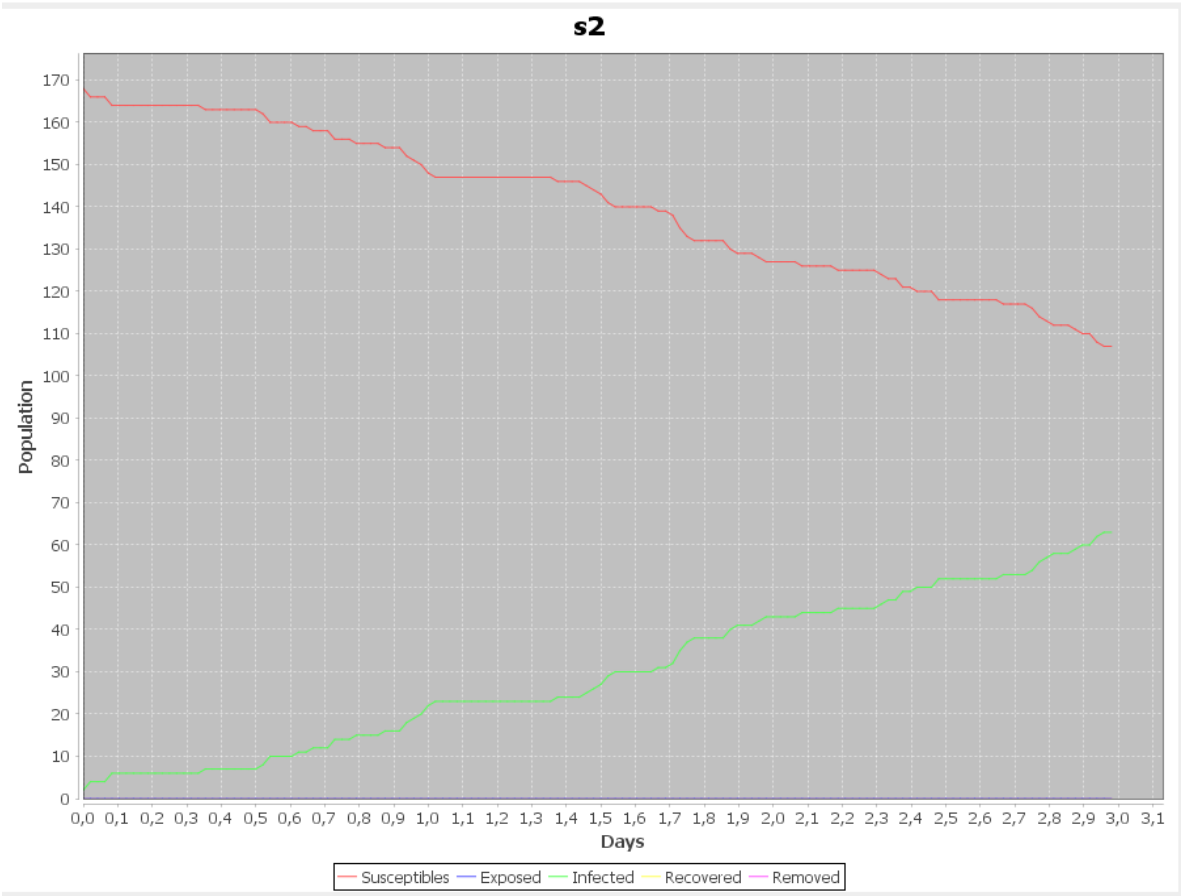

Ilustración 41 - Ventana Grafica de Simulación

4. Ver simulación (View simulation): Muestra el resultado de una simulación. Al hacer clic se abre en una ventana aparte como se puede ver en la ilustración 42.

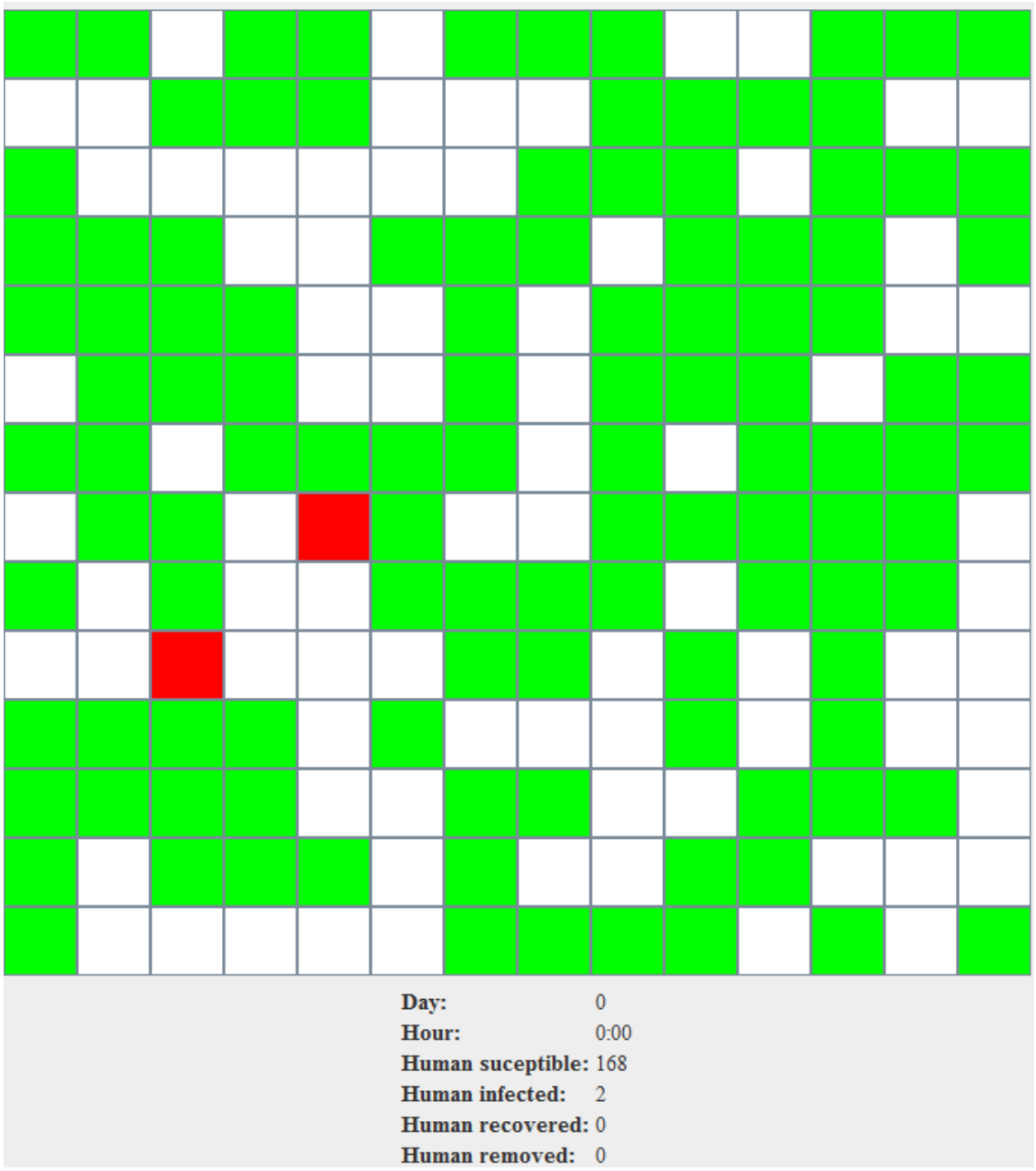

Ilustración 42 - Ventana Ver Simulación

5. Eliminar simulación (Remove): Este botón sirve para eliminar la simulación seleccionada. Al pulsarlo se mostrará un mensaje preguntando si está seguro que se desea borrar la simulación.

C) Manual de Usuario Aplicación Web.

Login

La página principal de la aplicación web es un formulario de ingreso a la misma. Para acceder a la aplicación hay que introducir el usuario y la contraseña. En el caso de que algún campo sea incorrecto, aparecerá un mensaje de error.

Ilustración 43 - Vista de Autenticación

Ilustración 44 - Autenticación Fallida

Página Principal

La vista principal de la aplicación, una vez el usuario se ha autenticado, es la creación de simulaciones que se puede ver en la ilustración 45.

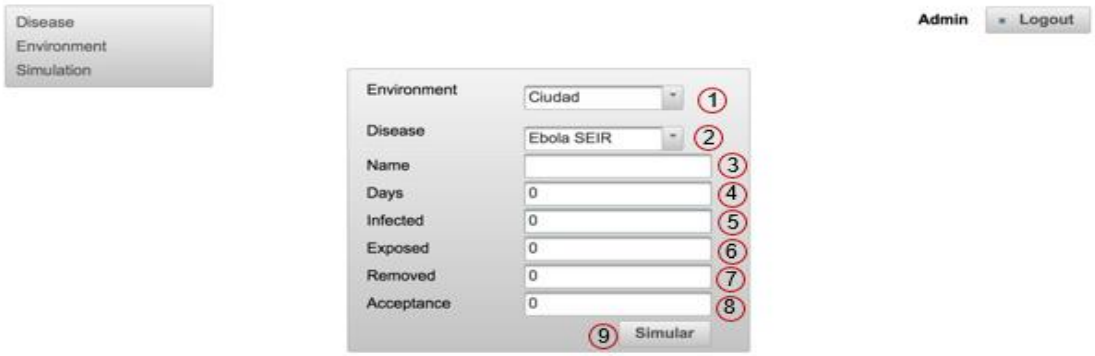

Ilustración 45 - Página Principal

1. Lista de entornos. En este campo se selecciona el entorno que se va a simular. Por defecto está seleccionado el primero de la lista.
2. Lista de enfermedades. En este campo se selecciona la enfermedad que se va a simular. Por defecto está seleccionada la primera de la lista.
3. Nombre de la simulación. El nombre no puede ser el mismo que el de una simulación ya existente. Además, el nombre tiene que tener entre 1 y 15 caracteres.
4. Número de días que dura la simulación. El número mínimo de días es 1, y el máximo se establece en función del número de habitantes del entorno seleccionado.
    - Entre 0 y 500 humanos. 170 días.
    - Entre 500 y 1000 humanos. 130 días.
    - Entre 1000 y 1500 humanos. 70 días.
    - Entre 1500 y 2000 humanos. 50 días.
5. Número de infectados iniciales. Este campo debe ser un número entre 0 y la población del entorno que se va a simular.
6. Número de personas en periodo de incubación iniciales. Este campo debe ser un número entre 0 y la población del entorno que se va a simular. Este campo sólo aparecerá si la enfermedad seleccionada es de tipo SEIR o SEIS.
7. Número de personas muertas iniciales. Este campo debe ser un número entre 0 y la población del entorno que se va a simular.
   La suma del campo 8, 9 y 10 no debe ser mayor que el número de habitantes del entorno seleccionado.
8. Aceptancia. Predisposición de los individuos de aceptar que están enfermos.
9. Botón de simulación. Al pulsar el botón comienza la simulación. Si alguno de los campos no es correcto, aparecerá un mensaje indicando los errores. Si hay una simulación ejecutándose en el servidor, aparecerá un mensaje indicándolo.

Página Disease

La página Disease (ilustración 46) permite crear, visualizar, editar y borrar una enfermedad.

Ilustración 46 - Página Disease

1. Lista de enfermedades creadas.
2. Nombre de la enfermedad. El nombre no puede ser el mismo que el nombre de una enfermedad ya existente. Además, el nombre tiene que tener 1 y 15 caracteres.
3. Tipo de enfermedad. Desplegable con los distintos tipos de enfermedad.
4. Índice de infectividad. Este campo debe ser un número decimal entre 0 y 1.
5. índice de mortalidad. Este campo debe ser un número decimal entre 0 y 1.
6. Distancia de infección. Este campo debe ser un número decimal entre 0 y la distancia de infección máxima definida por el usuario. La distancia de infección máxima puede ser definido por el administrador del sistema, por defecto es 100.
7. Número máximo de días que dura la infección. Este campo debe ser un número entre 1 y el número máximo de días que dura la infección. El número máximo de días que dura la infección puede ser definido por el administrador del sistema, por defecto es 100 días.
8. Número mínimo de días que dura la infección. Este campo debe ser un número entre 1 y el número máximo de días que dura la infección.
9. Número medio de días que dura la infección. Este campo debe ser un número entre el mínimo de días que dura la infección y el máximo de días que dura la infección.

10. Número máximo de días que dura el periodo de incubación. Este campo debe ser un número entre 1 y el máximo de días que dura el periodo de incubación. El número máximo de días que dura el periodo de incubación puede ser definido por el administrador del sistema, por defecto es 100 días. Este campo sólo se visualizará si el tipo de enfermedad es SEIR o SEIS.
11. Número mínimo de días que dura el periodo de incubación. Este campo debe ser un número entre 1 y el máximo de días que dura el periodo de incubación. Este campo sólo se visualizará si el tipo de enfermedad es SEIR o SEIS.
12. Número medio de días que dura el periodo de incubación. Este campo debe ser un número entre el número mínimo de días que dura el periodo de incubación y el número máximo de días que dura el periodo de incubación. Este campo sólo se visualizará si el tipo de enfermedad es SEIR o SEIS.
13. Botón para crear enfermedades. Al pulsar el botón se creará una enfermedad. Si el nombre de la enfermedad que se intenta crear ya existe, se mostrará un mensaje indicando que ese nombre ya existe.
14. Botón para editar una enfermedad. Al pulsar el botón se editará una enfermedad. Si el nuevo nombre de la enfermedad existe o hay una simulación relacionada con esa enfermedad, se mostrará un mensaje indicando que no se puede editar.
15. Botón para borrar una enfermedad. Al pulsar el botón se borrará la enfermedad. Si la enfermedad está relacionada con alguna simulación, se mostrará un mensaje indicando que no se puede borrar.

Página Environment

La página Environment permite crear, visualizar, editar y borrar un entorno.

Ilustración 47 - Página Environment

1. Lista de los entornos creados.
2. Nombre del entorno. El nombre no puede ser el mismo que el nombre de un entorno ya existente. Además, el nombre tiene que tener como mínimo un carácter y como máximo 15.
3. Número de habitantes. Este campo debe ser un número entre 1 y el número máximo de habitantes. El máximo puede ser definido por el administrador del sistema, por defecto es 2000 personas.
4. Área del entorno. Este campo debe ser un número decimal mayor que 0 y menor que el área máximo. El área máximo puede ser definido por el administrador del sistema, por defecto es 25 m^2.
5. Porcentaje de población de riesgo. Este campo debe ser un número entre 0 y 100.
6. Número de personas por casa. Este campo debe ser un número decimal entre 1 y la población del entorno.
7. Número de personas por trabajo. Este campo debe ser un número decimal entre 1 y la población del entorno.
8. Porcentaje de personas trabajando. Este campo debe ser un número entre 0 y 100.
9. Porcentaje de personas estudiando. Este campo debe ser un número entre 0 y 100.
   La suma del porcentaje de personas trabajando y de personas estudiando tiene que ser 100.
10. Nivel de desarrollo del entorno.
11. Botón para crear entornos. Al pulsar el botón se creará un entorno. Si el nombre del entorno que se intenta crear ya existe, se mostrará un mensaje indicando que ese nombre ya existe.
12. Botón para editar un entorno. Al pulsar el botón se editará un entorno. Si el nuevo nombre del entorno existe o hay una simulación relacionada con ese entorno, se mostrará un mensaje indicando que no se puede editar.
13. Botón para borrar un entorno. Al pulsar el botón se borrará el entorno. Si el entorno está relacionado con alguna simulación, se mostrará un mensaje indicando que no se puede borrar el entorno.

Página Simulation

La página Simulation (ilustración 48) muestra las simulaciones creadas y la información de cada una de ellas.

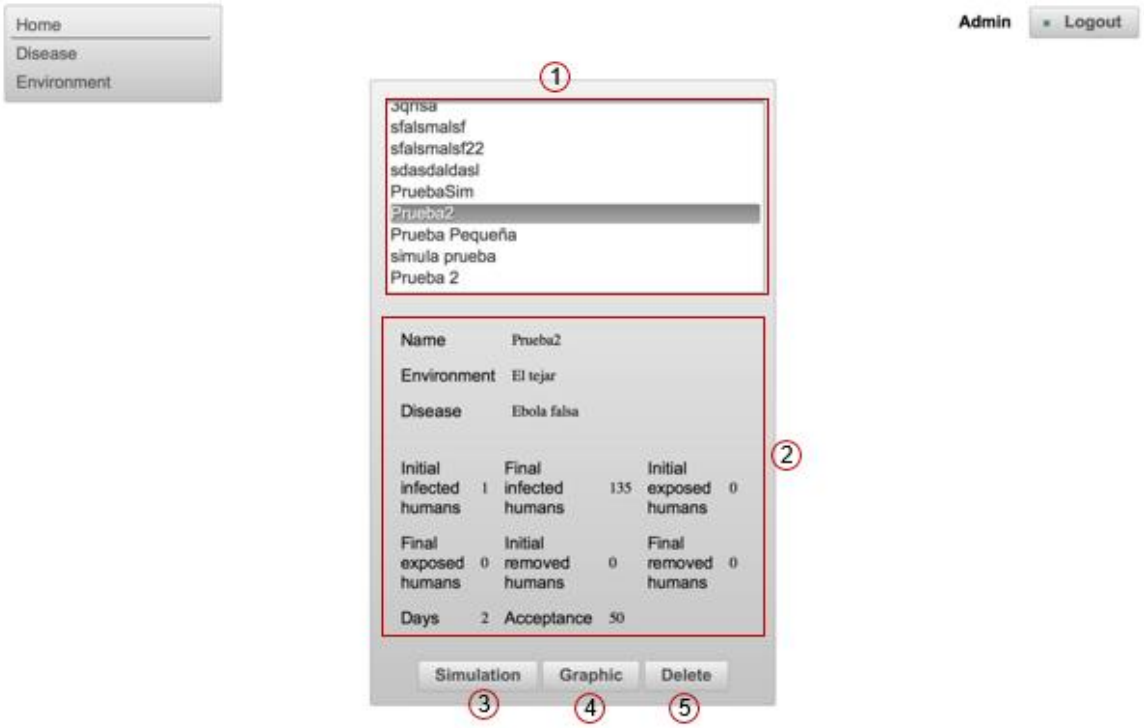

Ilustración 48 - Página Simulation

1. Lista de simulaciones.
2. Información acerca de la simulación seleccionada.
3. Botón para ver gráficamente la evolución de la simulación seleccionada.
4. Botón para ver una gráfica estadística con la evolución de la simulación seleccionada.
5. Botón para borrar la simulación seleccionada.

En esta página si se pulsa el botón Simulation se accederá a la siguiente vista (ilustración 49).

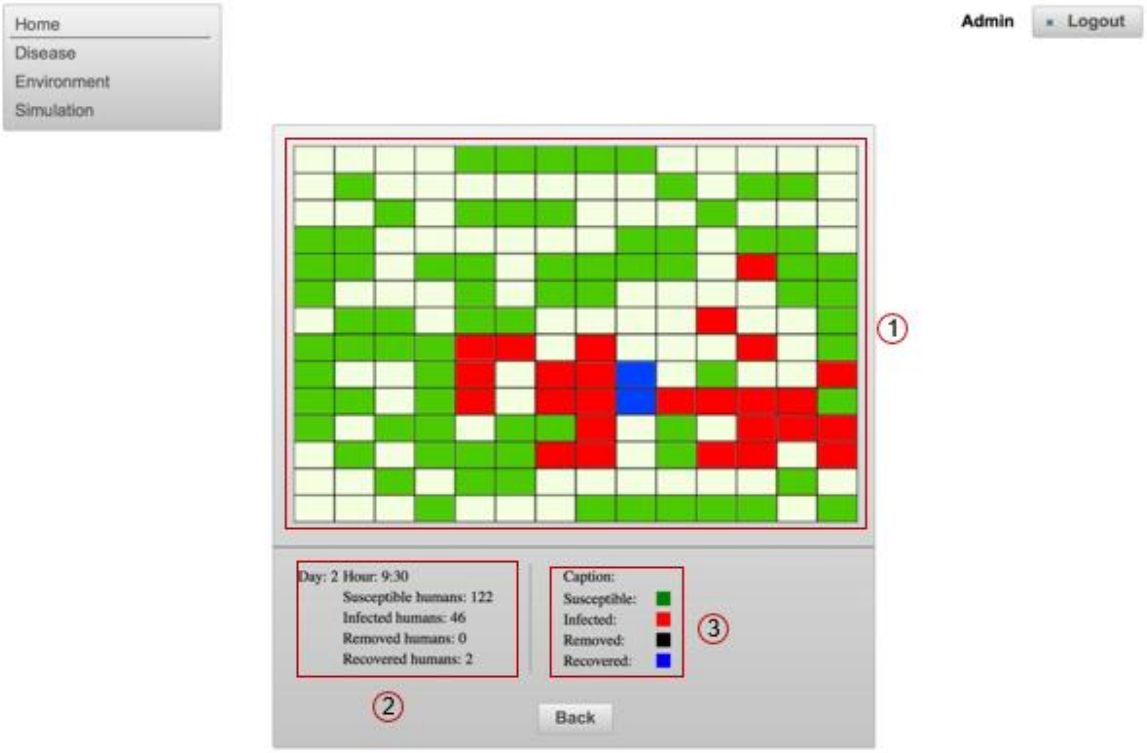

Ilustración 49 - Vista Gráfica de la Simulación

1. Cuadrícula que muestra la evolución de la simulación.
2. Información de la simulación en un instante.
3. Información sobre el significado de cada color.
4. Botón para volver a la página Simulation.
5. En la página Simulation si se pulsa el botón Graphic se accederá a la siguiente página (ilustración 50).

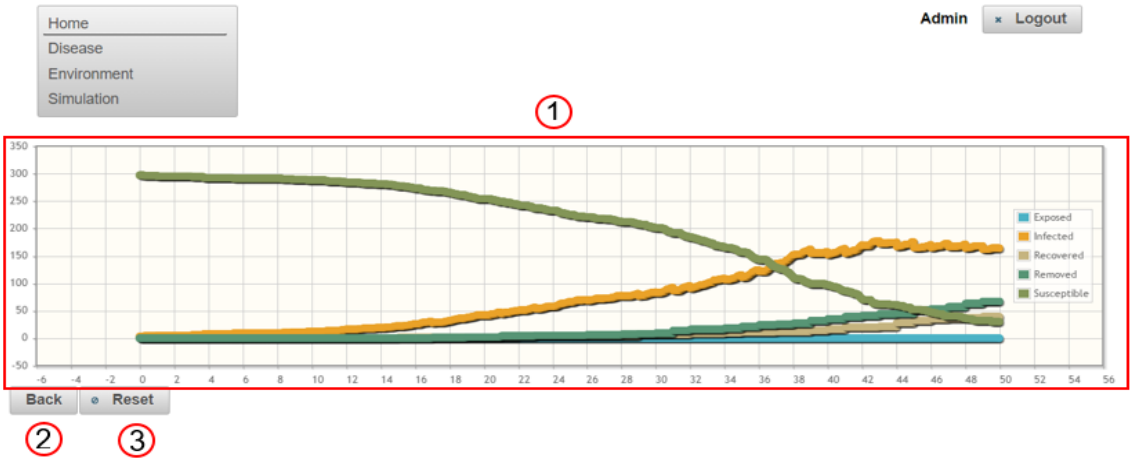

Ilustración 50 - Gráfica Estadística de la Simulación

1. Gráfica estadística que muestra la evolución de la simulación. Se puede hacer zoom sobre la gráfica. Para ello hay que pulsar sobre la gráfica y arrastrar sobre la parte que se quiere aumentar.
2. Botón para volver a la página Simulation.
3. Botón para volver al estado original de la gráfica estadística.

   Nota: Se puede hacer zoom sobre la gráfica al seleccionar un trozo de ella.

Barra de Navegación y Elementos Comunes

Como se puede observar en la ilustración 51, todas las páginas tienen varios elementos en común. Estos elementos se explican a continuación:

1. Menú de navegación. Este menú permite navegar a otras vistas diferentes de la actual.
2. Nombre de usuario.
3. Botón de cerrar sesión.

Ilustración 51 - Elementos de la Cabecera

NOTA: Al recargar la página, sea cual sea la página en la que se esté, la aplicación se dirigirá a la página de autenticación donde habrá que autenticarse de nuevo.

D) Glosario

- **AFL** (Academic Free License): Licencia de software libre permisiva creada en 2002.
- **AID** (Agent IDentifier): Identificador de una agente en la plataforma JADE.
- **AJAX** (Asynchronous JavaScript And XML): Técnica que permite realizar llamadas asíncronas al servidor HTTP evitando recargar toda la página.
- **AMS** (Agent Management System): Es un agente de la plataforma JADE que se encarga de gestionar el funcionamiento de la de la misma.
- **API** (Application Programming Interface): Conjunto de funciones y procedimientos que ofrece una biblioteca para ser utilizado por otro software.
- **BSD** (Berkeley Software Distribution): Es una licencia de software permisiva otorgada principalmente para los sistemas BSD.
- **CRUD** (Create, Read, Update and Delete): Funciones elementales en bases de datos o la capa de persistencia en un software.
- **CSS** (Cascading Style Sheets): Lenguaje utilizado para definir la presentación de un documento escrito en HTML o XML.
- **DF** (Directory Facilitator): Componente de la plataforma JADE que mantiene una lista exacta, completa y actualizada de los servicios prestados por los agentes.
- **FIPA** (Foundation for Intelligent Physical Agents): Organismo que desarrolla y establece estándares softwares para los sistemas basados en agentes.
- **GUI** (Graphical User Interface): software que actúa de interfaz de usuario, utilizando distintos elementos gráficos.
- **HTML** (HyperText Markup Language): Estándar que sirve de referencia para la elaboración de páginas web, en el cual se define una estructura y un código.
- **HTTP** (Hypertext Transfer Protocol): Protocolo usado en las transacciones de la World Wide Web.
- **IA** (Inteligencia Artificial)**:** es un área multidisciplinaria, que estudia la creación y diseño de sistemas capaces de resolver problemas cotidianos por sí mismos.
- **JADE** (Java Agent DEvelopment Framework): Es un software para el desarrollo de agentes, implementado en Java.
- **JDBC** (Java Database Connectivity): Es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.
- **JPA** (Java Persistence API): Es una API de persistencia desarrollada para la plataforma Java EE.
- **JSF** (JavaServer Faces): Tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

- **JSON** (JavaScript Object Notation): Es un formato ligero para el intercambio de datos.
- **JSP** (JavaServer Pages): Tecnología para crear páginas web dinámicas usando el lenguaje de programación Java.
- **KIF** (Knowledge Interchange Format): Es un lenguaje de programación orientada para el intercambio de conocimientos entre programas.
- **KQML** (Knowledge Query and Manipulation Language): Es un lenguaje y un protocolo para la comunicación entre agentes software y sistemas basados en el conocimiento.
- **GNU LGPL** (GNU Lesser General Public License): Es una licencia de software creada por la *Free Software Foundation* que pretende garantizar la libertad de compartir y modificar el software.
- **ORDBMS** (Object-Relational Database Management System): Es una extensión de la base de datos relacional tradicional, a la que se le proporcionan características de la programación orientada a objetos.
- **POJO** (Plain Old Java Object): Denominación utilizada por programadores Java para enfatizar el uso de clases simples.
- **RDBMS** (Relational DataBase Management System): Es un gestor de base de datos basado en el modelo relacional.
- **OS** (Operating System): Es un programa o conjunto de programas de un sistema informático encargado de gestionar los recursos de hardware y proveer de distintos servicios a los programas de aplicación.
- **URI** (Uniform Resource Identifier): Cadena de caracteres que identifica los recursos de una red de forma unívoca.
- **WADL** (Web Application Description Language): Es una descripción XML de un servicio REST.
- **XML** (eXtensible Markup Language): Es un lenguaje de marcas utilizado para almacenar datos en forma legible.