ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

SISTEMA MULTIAGENTE PARA LA SIMULACIÓN DE EPIDEMIAS

MULTIAGENT SYSTEM FOR EPIDEMIC SIMULATION

Realizado por
MANUEL DAVID PEÑARANDA VELA
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: El presente trabajo consiste en elaborar un sistema que permita simular epidemias en un entorno a través de agentes que representan a los habitantes del entorno simulado. El trabajo consta de cuatro partes: una aplicación web realizada en JSF, una aplicación de escritorio realizado en Java, un sistema multiagente, que se encarga de realizar la simulación, realizado en Java junto al framework JADE y un servidor web que contiene la aplicación web y el sistema multiagente. La simulación, el entorno y la enfermedad pueden ser configuradas, por parte del usuario, con distintos parámetros necesarios para la realización de la simulación. Una vez realizada la simulación, ésta puede ser visualizada a través de una animación y/o a través de un gráfico que representa la evolución de la simulación. Con el fin de que el sistema tuviera un funcionamiento óptimo, se han desarrollado pruebas de estrés aumentando el número de días y de personas para poder comprobar la solidez del sistema y así realizar mejoras si es necesario. Todo esto conforma un sistema cuya finalidad es obtener unos datos a partir de los cuales se pueden realizar distintos estudios y sacar conclusiones a partir de ellos, ayudando a investigar cómo se comporta una epidemia en unas determinadas condiciones y también distintas formas de poder combatirlas.

Palabras claves: multiagente, epidemia, simulación, JADE, JSF, enfermedad, agente, java, aplicación web, aplicación escritorio, servicios REST

Abstract: The present work is to develop a system to simulate epidemics in an environment through agents who represent the people of the simulated environment. The work consists of four parts: a web application developed with JSF, a desktop application developed with Java, the simulation system developed with Java and the JADE framework and a web server which uses REST services to return the results of the simulations. The simulation, the environment and the disease can be configured by the user, with different parameters required for performing the simulation. After the simulation, it can be viewed through an animation and/or through a chart representing the evolution of the simulation. Stress tests has been developed to optimize the

performance of the system. This stress tests consists in increasing the number of days and people in order to test the robustness of the system and thus make improvements if necessary. All this forms a system whose purpose is to obtain data from which they can conduct studies and draw conclusions from them, helping investigate how an epidemic behaves under certain conditions and also different ways to combat them.

Keywords: multiagent, epidemic, simulation, JADE, JSF, disease, agent, java, web aplication, desktop aplication, REST servicies

Índice

Índice de Ilustraciones

Índice de Tablas

1. Introducción

1.1. Antecedentes

La epidemia de ébola de 2014 que afectó mayormente a la población africana, activó la alarma social ante el miedo de una expansión mundial de una enfermedad con una tasa de mortalidad elevada. Este brote comenzó en África, cuyos niveles de desarrollo son muy bajos y, por tanto, la posibilidad de contagio es mayor. El miedo de que esta enfermedad se propagase a otros continentes debido a la repatriación de voluntarios contagiados o de personas expuestas que viajaran hacia otro país, puso de manifiesto la necesidad de establecer medidas de control y seguimiento ante este tipo de epidemias para minimizar daños, evitando así una posible pandemia e impedir otra epidemia en el futuro.

Como consecuencia de estos hechos, era necesario disponer de un mecanismo que permitiera estudiar qué impacto tendrá una enfermedad de este tipo en una determinada población y cómo actuar ante ella de la forma más adecuada para minimizar su efecto usando los recursos disponibles.

1.2. Descripción general

Es innegable que la evolución constante de la informática ha sido útil en muchos sentidos, como puede ser la vida cotidiana, la enseñanza, la investigación, etc. Esto ha permitido que un usuario medio pueda utilizar las aplicaciones de manera más fácil, que un profesor pueda transmitir la materia a sus alumnos de manera más completa o que un investigador pueda realizar sus estudios de una forma más sencilla, efectiva y ágil. Por otro lado, y en parte como consecuencia de esta evolución informática, también ha habido una evolución de la inteligencia artificial cuyo objetivo ha sido desarrollar sistemas cuya misión es resolver problemas cotidianos de manera autosuficiente.

Como respuesta a la necesidad planteada debido a la epidemia del ébola, y aprovechando las ventajas de la informática y de la inteligencia artificial, surge la idea de modelar un sistema informático que, a través de los sistemas multiagente, simule el comportamiento de una población real ante una enfermedad en un determinado entorno.

Las ventajas que proporciona realizar un sistema informático para satisfacer esta necesidad son las siguientes:

- **Agilidad.** El usuario del sistema agiliza el estudio ahorrando tiempo de cálculos, puesto que la aplicación realiza todo el cálculo y los presenta de manera fácil de entender.

- **Ahorro.** Al agilizarse los estudios se ahorra tiempo y por tanto recursos necesarios para la realización del estudio.
- **Centralización de la información.** Reúne en un único sitio toda la información pudiendo ser consultada por el usuario en todo momento.
- **Integración.** La aplicación puede ser instalada en servidores.
- **Fiabilidad.** Los cálculos realizados son fiables y al utilizar la inteligencia artificial se puede simular de manera más fiable una población.
- **Portabilidad.** Se puede acceder a la aplicación de manera independiente al sistema en el que esté instalado.
- **Recursos.** La ejecución de la aplicación no requiere demasiados recursos en el equipo del usuario.

1.3. Objetivos

El objetivo del presente Trabajo de Fin de Grado es desarrollar un sistema informático que, mediante un sistema multiagente, se simule una enfermedad en una población situada en un entorno con unas determinadas condiciones definidas por el usuario.

Para que se puedan aprovechar las ventajas descritas anteriormente, se realizará una aplicación web y una de escritorio que permita una fácil portabilidad. Así como, un servidor web donde se hará el procesamiento más costoso para evitar así consumir demasiados recursos en el equipo del usuario. Además el sistema permitirá un ajuste de la población, del entorno y de la enfermedad para que el estudio sea lo más fidedigno posible a la realidad.

Aunque puede ser utilizado por todo tipo de usuarios, dos son los principales demandantes. Por lo tanto, el sistema debe de realizar las funciones que cubran las necesidades de estos principales tipos de usuario.

Para el **investigador**:

- Investigar comportamiento de la enfermedad en distintas poblaciones y detectar patrones de esa enfermedad.
- Con la ayuda de estas investigaciones, desarrollar vacunas o medicamentos para combatir estas enfermedades.
- Asesorar para evitar posibles epidemias en el futuro.
- Estudiar el impacto de una enfermedad en una determinada población.

Para los **gobiernos y organismos como la OMS**:

- Ayudar a tomar decisiones para minimizar daños en un país frente a una determinada enfermedad y paliar sus efectos en la población.
- Establecer medidas de control y seguridad apropiadas para evitar su expansión.
- Mejorar infraestructuras, protocolos y la sanidad para evitar posibles epidemias.

Para realizar este proyecto se ha marcado también el objetivo de usar tecnologías Open Source, debido a que no hay otro tipo de opciones que permitan numerosas ventajas como ahorro en coste de licencias, uso de estándares abiertos y soporte por parte de las comunidades de los mismos.

1.4. Estructura de la memoria

A continuación se realizará una introducción de los capítulos de los que va a constar esta memoria:

Capítulo 2. Contexto del trabajo

En este capítulo se explicará el contexto del trabajo. Se hará un acercamiento a la teoría de los agentes y se hablará sobre los distintos tipos de enfermedades.

Capítulo 3. Tecnologías y herramientas

Se realiza un estudio de las tecnologías y herramientas existentes para el proyecto y se comparan para elegir la más adecuada, de entre todas las posibles, para usarlas en el desarrollo del proyecto.

Capítulo 4. Especificación y análisis

Se estudia la metodología de desarrollo más idónea para este tipo de proyecto y se elabora una lista de requisitos funcionales y no funcionales de las aplicaciones, detallando, además, los casos de usos extraídos a partir de estos requisitos. También se especifican los requisitos de la simulación y se analiza la interfaz de usuario y la base de datos.

Capítulo 5. Diseño

En este capítulo se especifican los módulos principales que componen el sistema y los patrones de diseños utilizados. Esta especificación viene acompañada de distintos diagramas de UML que facilita la explicación. También se explica el modelo relacional de la base de datos.

Capítulo 6. Implementación y Pruebas

Se realiza la explicación de la implementación de ciertas partes del sistema, la explicación de estas partes se debe a su complejidad o a su interesante función dentro del sistema. Posteriormente, se comenta los problemas más importantes a la hora de realizar la implementación. Por último, se detallan las principales pruebas ejecutadas en el proyecto y se explica las conclusiones sacadas a partir de las distintas pruebas.

Capítulo 7. Conclusiones y mejoras futuras

En este último capítulo se detallan las conclusiones del proyecto y los resultados a los que se han llegado tras desarrollarlo. También se puntualiza distintas mejoras que puede tener el sistema en versiones futuras.

2. Contexto del trabajo

2.1. Introducción

En este capítulo se va a abordar todo lo relacionado con los agentes: qué son, su arquitectura, cómo se comunican, cómo implementarlos y sus aplicaciones. También se explicará los distintos tipos de enfermedades.

2.2. Teoría de agentes

El término agente es un término muy rico en cuanto a significado. Hay muchos tipos de agentes, sólo hace falta buscar el término “Agente” en Wikipedia para darse cuenta de ello, desde agentes gramaticales a agentes computacionales pasando por agente de negocios, policial, etc. En el contexto de este proyecto, la palabra agente se refiere a una entidad inteligente con un comportamiento humano.

A partir de estos agentes nace un nuevo paradigma de programación, la programación orientada a agentes (AOP). Este paradigma mezcla conceptos de la teoría de la inteligencia artificial y de los sistemas distribuidos, modelando agentes con capacidades como la autonomía, la proactividad y la habilidad de comunicarse. Utilizar este paradigma a la hora de desarrollar proporciona numerosas ventajas:

- **Independencia.** Pueden realizar tareas complejas de manera independiente y sin necesidad de la interacción del usuario.
- **Integración.** Se integran y son compatibles con otras tecnologías como las aplicaciones web o las bases de datos.
- **Mejora la calidad del producto desarrollado y su funcionalidad.** Los sistemas basados en agentes son intuitivos y sencillos de utilizar, ya que son flexibles y se adaptan al usuario.
- **Menor coste.** Al ser reutilizable requieren menos recursos y reduce el tiempo de desarrollo.
- **Fácil transformación y evolución.** Modificando el conocimiento u los objetivos de los agentes es posible cambiar la funcionalidad rápidamente.

2.2.1. ¿Qué es un agente?

El término agente es un término muy usado dentro de las ciencias de la computación: inteligencia artificial, bases de datos o sistemas operativos entre otros. No hay un consenso sobre una definición exacta sobre este término, una de las definiciones es la siguiente: “sistema hardware o software que funciona de forma autónoma y puede interactuar con su entorno y con otros agentes”. Otra de las definiciones, que suele ser citada, es la de Wooldridge que define un agente de la siguiente manera: “un agente es un sistema informático situado en un entorno y que es capaz de realizar

acciones de forma autónoma para conseguir sus objetivos de diseño". Todas las definiciones sobre agente están de acuerdo en que un agente es una entidad software adaptable, es decir, puede funcionar en distintos entornos o plataformas. Es capaz de realizar una serie de objetivos de manera autónoma y racional maximizando el resultado esperado. Estos objetivos pueden ser realizados de manera individual intercambiando información con el entorno, junto a otros agentes humanos o junto a otros agentes computacionales, la colaboración con éstos últimos conforma un **sistema multiagente**. Los sistemas multiagente pueden modelar sistemas complejos destinados a resolver problemas que son difíciles de resolver por un agente individual y lograr entre varios agentes un objetivo común. Los agentes que conforman el sistema pueden interactuar con otros agentes de manera directa o indirecta, cooperando entre sí en beneficio mutuo o para luchar por sus propios intereses.

Por tanto, un agente es autónomo porque actúan sin necesidad de interacción humana; social, porque puede comunicarse en pos de conseguir un objetivo; reactivo, ya que responde a cambios en el entorno; proactivo, debido a que tiene iniciativa y es capaz de anticiparse a los problemas; móvil, puesto que puede viajar entre distintos nodos y además, es racional y aprende de las experiencias. Estos aspectos esenciales hacen que se puedan comportar de manera inteligente.

2.2.2. Arquitectura

La arquitectura de agentes determina los mecanismos necesarios para que un agente tenga un comportamiento adecuado. El objetivo de una arquitectura es especificar cómo se descomponen los agentes en distintos módulos que actúan entre sí para lograr la funcionalidad necesitada. Hay cuatro principales tipos de arquitectura de agentes: deliberativas, reactivas y arquitectura en capas o híbridas.

- **Deliberativas.** Se basa en la representación simbólica del conocimiento. El entorno es representado y manipulado simbólicamente usando mecanismos de razonamiento. La ventaja de usar esta arquitectura es su facilidad a la hora de codificar, ya que el conocimiento es simbólico. Por el contrario, es complicado conseguir traducir el mundo real en un simbolismo adecuado. Una arquitectura deliberativa es BDI (Belief, Desire, Intention). Esta arquitectura es la más popular entre todas las arquitecturas de agentes, debido a que combina un modelo filosófico del razonamiento humano cuya comprensión no es difícil, tiene bastantes implementaciones y una semántica lógica elegante y abstracta. BDI significa creencia, deseo e intención; donde creencia es la información que tiene un agente sobre el entorno; el deseo es el objetivo que debe cumplir el agente, y las intenciones son los deseos que el agente debe perseguir.
- **Reactivas.** Implementan una toma de decisión en el que se realiza una acción a partir de una situación que se percibe a través de un estímulo. La arquitectura más conocida es la de subsunción (Brooks, 1991). Las ideas claves de esta

arquitectura es que no hace falta construir un modelo simbólico para generar comportamientos inteligentes, ya que se pueden generar a partir de sistemas complejos.

- **Híbridas.** Estas arquitecturas combina los aspectos de los modelos deliberativos y reactivos. Para ello se implementa una arquitectura en capas que consta de dos tipos: horizontal, donde todas las capas están conectadas a los sensores y actuadores, y vertical, donde sólo una capa está conectada a los sensores y actuadores. Estas capas se organizan de manera jerárquica dividiéndose, normalmente, en tres niveles:
    - *Reactivo*, el nivel más bajo. Se toman decisiones a partir de los estímulos recibidos.
    - *Conocimiento*, el nivel intermedio. Conocimiento que tiene el agente del medio con la ayuda de una representación simbólica.
    - *Social*, el nivel más alto. Se maniobran los aspectos sociales del entorno.

Otra arquitectura más relacionada con los sistemas multiagente es la arquitectura FIPA. En las especificaciones de FIPA se definen ciertas características que tienen que cumplir las plataformas de gestión de sistemas multiagente. Estas plataformas parten de un núcleo, la plataforma de agentes propiamente dicha, que proporciona la infraestructura necesaria para el uso y el desarrollo de agentes. Estas plataformas proporcionan una serie de servicios que son los siguientes:

- **Sistema de Gestión de Agentes.** El AMS se encarga de realizar toda la gestión principal, conociendo en todo momento el estado de la plataforma y los agentes pertenecientes a ella. El AMS también proporciona un servicio de nombres donde se asocia un nombre a cada agente.
- **Facilitador de Directorio.** Es un servicio de páginas amarillas donde los agentes se pueden registrar y también pueden buscar por capacidades a otros agentes registrados en el directorio.
- **Sistema de Transporte de Mensajes.** El STM se encarga de gestionar el envío de mensajes entre los agentes de una plataforma y entre los agentes de distintas plataformas.

2.2.3. Comunicación entre agentes

La comunicación en un sistema multiagente es esencial, debido a que los agentes tienen que ser capaces de comunicarse con los usuarios, los recursos del sistema u otros agentes con el fin de coordinarse para lograr un objetivo. El primer lenguaje de comunicación fue KQML desarrollado a principio de los 90. Este lenguaje permite intercambiar información definida a través de una serie de verbos performativos, el

contenido del mensaje es representado a través de KIF que es una implementación lógica de predicados de primero orden.

Aunque actualmente el lenguaje más utilizado es FIPA ACL que incorpora algunos aspectos de KQML. FIPA ACL posibilita usar diferentes lenguajes para definir el contenido, y proporciona una serie de protocolos para gestionar los mensajes. Estos protocolos nacen a partir de patrones típicos que se encuentran en las estructuras de conversación, algunos de estos protocolos son:

- *Request.* Cuando se le pide a un agente que realice cierta acción.
- *Subscribe*. En el que un agente es notificado si una condición se cumple.
- *Contract net.* Un agente delega la realización de una tarea a un agente. Para ello pide la realización de esta tarea a un conjunto de agentes que responden con una propuesta. El protocolo acaba con la elección de un agente de ese conjunto que será quien realice la tarea.

2.2.4. Lenguajes de programación y herramientas

Los lenguajes de programación más adecuados para desarrollar un sistema multiagente son los lenguajes de programación orientado objetos, ya que el concepto de agente es muy similar al concepto de objeto, puesto que comparten algunas propiedades. No obstante, éstos difieren en ciertas características como la autonomía.

Un lenguaje de programación orientado a agentes debe incluir las estructuras necesarias para definir un agente y facilitar mecanismos que permita a un agente tener objetivos, planes, roles o normas.

También existen plataformas de agentes o frameworks que permiten el desarrollo de sistemas multiagente. Los más conocidos son los siguientes:

- **JADE.** Es la implementación más extendida del estándar FIPA cuyo desarrollo fue comenzado por Telecom Italia a finales de 1998. Este framework es un middleware escrito en Java que proporciona*:*
  - Una plataforma para ejecutar agentes JADE.
  - Librerías para crear agentes mediante herencia y definir comportamientos personalizados o redefinirlos.
  - Herramienta gráfica que permite monitorizar y gestionar los agentes y la plataforma*.*

  Los agentes de JADE se comunican a través de mensajes ACL y tienen un ciclo de vida con estados como iniciado, activo, suspendido, borrado, etc.

- **MaDKit.** Es una librería Java para diseñar y simular sistemas multiagente. Es una plataforma personalizable y escalable de propósito general. Como soporte tiene documentación, foro y ejemplos.

- **MASON.** Es una librería Java desarrollada por la Universidad de George Mason. Está diseñado para ser utilizado en sistemas multiagente cuyo objetivo es lograr una simulación, conteniendo librerías para hacer modelos y una herramienta gráfica para visualizar la simulación en 2D y 3D. Como soporte tiene documentación, tutoriales, extensiones de terceros y artículos de investigación que referencian a esta herramienta.
- **NetLogo.** Es un lenguaje de programación orientado a agentes y también un entorno de desarrollo. Está orientado a ser usado por los docentes para enseñar conceptos de programación o para usuarios con bajos conocimientos de programación. El entorno tiene incluido una librería de modelos de ejemplos que simulan distintos tipos de fenómenos. Esta opción no es tan potente como otras opciones, debido a que está orientado más a la enseñanza que a la investigación. La página web también tiene tutoriales, documentación y extensiones de terceros.
- **Repast.** Repast puede ser utilizado con varios lenguajes como Java, Python, C++, etc. Es una plataforma para modelar y simular agentes. Tiene una amplia variedad de agentes y de ejemplos, además de permitir a los usuarios modificar a los agentes dinámicamente en tiempo de ejecución. También incluye librerías para algoritmos genéticos y redes neuronales. Tiene soporte en forma de documentación, tutoriales y ejemplos.

2.2.5. Aplicaciones de los Sistemas Multiagente

Los sistemas multiagente tienen multitud de aplicaciones. Como consecuencia de ello, están teniendo una contribución importante a la hora de resolver problemas en diversos campos como la medicina, el comercio electrónico, las telecomunicaciones, etc.

En las industrias se pueden aplicar para controlar los procesos, planificar la fabricación, diagnosticar sistemas o logística de transporte. Con ello se puede gestionar de manera inteligente los procesos de fabricación, la ruta óptima de entrega, o ser notificado si un sistema no funciona correctamente evitando estar en constante atención para verificar el funcionamiento.

En la gestión del tráfico y del transporte es usado también. Por ejemplo para controlar el tráfico aéreo, en este caso cada vez que entra un avión al espacio aéreo de un determinado aeropuerto se le asigna un agente, proporcionándole toda la información necesaria para que puedan gestionar el aterrizaje del avión de forma adecuada.

Otra aplicación puede ser encontrado en la medicina donde se han desarrollado aplicaciones para solucionar diferentes problemas relacionados con la gestión de los pacientes.

A la hora de construir edificios o espacios también se puede aplicar estos sistemas, simular como va a funcionar el espacio una vez esté en uso es útil a la hora de diseñar el espacio, ya que el espacio tiene que estar preparado para soportar distintas situaciones como una posible evacuación de emergencia y esto puede ser simulado a través de un sistema multiagente.

2.3. Modelo de enfermedades

Hay cuatro principales modelos matemáticos de enfermedades:

- **SIS.** En este modelo hay tres estado: *susceptible* (sano), *infected* (infectado) y *susceptible* (sano). Este modelo representa el caso en el que una persona se contagia y, una vez recuperada, puede volver a contraer esa enfermedad.
- **SIR.** En este modelo hay tres estados: *susceptible* (sano), *infected* (infectado) y *recovered* (recuperado). La diferencia de este modelo con respecto a SIS es que una vez se ha recuperado la persona, ésta se vuelve inmune a la enfermedad.
- **SEIS y SEIR.** Estos modelos son iguales a SIS y SEIR respectivamente. En este caso, se añade un nuevo estado, *exposed* (expuesto), que representa el periodo de incubación de la enfermedad. Durante este periodo, la persona expuesta no está en condición de infectar a otras personas.

3. Tecnologías y herramientas

3.1. Repositorios

El uso de repositorios faculta centralizar el proyecto que se está desarrollando en un servicio de alojamiento. Esto permite trabajar en equipo de manera más eficiente permitiendo mantener un control de versiones que facilita el mantenimiento. Para el control de versiones se suele utilizar Git, Subversion, Mercurial o CVS. De ellos se realizó un estudio para determinar cuál era el más adecuado:

- **Git:**
  - Permite una rápida gestión de ramas y mezcla de diferentes versiones.
  - La gestión es distribuida.
  - Puede emular CVS, utilizar los repositorios Subversion y conectar con el repositorio por HTTP, FTP o SSH.
  - Gestión eficiente de proyectos grandes.
  - Si hay una modificación posterior a una versión se notifica de la existencia de ese cambio.
- **Subversion:**
  - Creación de ramas eficientes.
  - Sólo se envían al servidor y reciben del servidor las diferencias.
  - Manejo eficiente de archivos binarios.
  - No facilita tener total conocimiento de los cambios que se han realizado.
- **Mercurial:**
  - Utiliza los protocolos SSH y HTTP para sincronizar los archivos.
  - Esta sincronización es eficiente respecto al uso de CPU y ancho de banda.
  - Dispone de interfaz web.
- **CVS:**
  - Primer sistema de control de versiones de código abierto.
  - Sólo versiona ficheros.
  - No maneja muy bien ficheros grandes ni ficheros binarios.
  - Buen rendimiento en el lado del cliente.

Debido al popular uso de Git y a que se tenía experiencia con esta tecnología, se llegó a la conclusión de que era la mejor opción. Como servicio de alojamiento se eligió Bitbucket frente a GitHub dado que eran similares, pero Bitbucket tenía la ventaja de que en el plan gratuito se podía tener repositorios privados.

3.2. Servidor

Dado que una parte del proyecto es una aplicación web que debe estar integrado en un servidor de aplicaciones, era necesario encontrar un servidor de aplicaciones adecuado. Había dos opciones: GlassFish y Apache TomEE. El servidor GlassFish era conocido, ya que se había utilizado antes pero, dado las malas experiencias, se

sugirió la idea de utilizar Apache TomEE. Sin embargo, debido a varios problemas para poder lanzar aplicaciones en este servidor, se tuvo que volver a utilizar GlassFish. La versión de GlassFish utilizada es la 4.1.

3.3. Bases de datos

Este proyecto requiere de muchos datos y, por lo tanto, de muchas operaciones de acceso de escritura y lectura a la base de datos. Dado que hay un amplio abanico de opciones, había que realizar una selección entre esas opciones, por tanto se evalúo tres opciones: MySQL, Oracle y PostgreSQL. Se eligieron estas tres opciones, ya que se había trabajado anteriormente con MySQL y Oracle, y PostgreSQL es una plataforma Open Source que resulta bastante interesante. En la Tabla 1 se puede encontrar una comparación de los tres gestores de bases de datos.

	Licencia	SO	Tipo	Rendimiento
MySQL	GLP	Multiplataforma	RDBMS	Alto
Oracle	Privativa	Multiplataforma	ORDBMS	Alto
PostgreSQL	BSD	Multiplataforma	ORDBMS	Medio

Tabla 1 - Comparación de gestores de base de datos

Aunque Oracle es más potente que MySQL, el hecho de que su licencia sea privativa y que la diferencia de potencia no sea destacable, hizo que se decidiera utilizar MySQL por su alto rendimiento y su licencia GLP. Para crear la base de datos y gestionarla se ha utilizado MySQL Workbench, y el motor de la base de datos es Inno DB.

3.4. Plataforma de Sistema Multiagente

El núcleo de este proyecto es el desarrollo de un sistema multiagente, por lo que era necesario seleccionar una plataforma donde poder desarrollar este sistema. Se barajó la posibilidad de utilizar una de las herramientas descritas en el Capítulo 2.5. Netlogo se descartó, debido a que se requería algo más potente. Por lo tanto, se evalúo las siguientes plataformas:

	Lenguaje de programación	Curva de aprendizaje	Soporte
Repast	Java, Python, etc.	Media	Medio
Mason	Java	Media	Alto
MadKit	Java	Baja	Medio
JADE	Java	Baja	Alto

Tabla 2 - Comparación de plataformas de Sistemas Multiagente

En un principio se pensó realizar el proyecto en Mason, pero tras una evaluación más exhaustiva se decidió utilizar JADE, ya que es de las plataformas que mejor cumplen con el estándar FIPA y las prestaciones que ofrece ayudaban a la realización del proyecto, otra razón por la que se inclinó hacia JADE fue que se disponía de bastante soporte.

3.5. Tecnologías usadas para el back-end

Para el back-end se ha utilizado el lenguaje de programación Java para la programación del sistema multiagente, debido a que el framework JADE utiliza ese lenguaje. Para la aplicación web se ha utilizado Java EE, que es una plataforma abierta para desarrollar y desplegar aplicaciones empresariales en entornos distribuidos. Éstas parten de la plataforma Java por lo que las aplicaciones desarrolladas son aplicaciones Java. Java EE tienen un componente llamado EJB que es usado para ofrecer funcionalidades a través de interfaces y de una manera homogénea a las aplicaciones clientes

3.6. Tecnologías usadas para el front-end

En el lado del cliente se ha utilizado JSF para construir las vistas. JSF simplifica el desarrollo de las interfaces de usuario incluyendo componentes de interfaz de usuario, de validación, gestión de eventos, soporte para internacionalización, etc. Las páginas creadas con JSF usan JSP, que se encarga de traducir el código a HTML. Se ha utilizado la versión 2.2 de JSF para el desarrollo de las vistas.

Para ciertas funcionalidades, como la simulación gráfica, se ha tenido que utilizar JavaScript para poder desarrollar la lógica en el lado del cliente. También se ha utilizado AJAX permitiendo interactuar con el servidor sin tener que recargar la página.

Para dar estilo a las interfaces se ha aplicado CSS y también PrimeFaces. Éstos dan un toque más vistoso a las páginas y añaden funcionalidades como la comunicación entre un bean y JavaScript. Se ha utilizado la versión 5.2 de PrimerFaces.

En la versión de escritorio de la aplicación se ha utilizado el lenguaje de programación Java, puesto que permite ser multiplataforma al ser portable. La aplicación de escritorio hace uso de servicios webs, en concreto servicios RESTful, para comunicarse con la base de datos.

3.7. Entorno de desarrollo

Para desarrollar el proyecto se ha utilizado el entorno de desarrollo Netbeans. Este entorno, desarrollado por Oracle, permite desarrollar todo tipo de aplicaciones Java, desde una aplicación Java hasta una aplicación Java EE. También permite trabajar

con servicios web y lleva integrado el servidor de aplicaciones GlassFish, lo que lo hace un entorno de desarrollo idóneo para este proyecto.

4. Especificación y análisis

4.1. Metodología de desarrollo

Es importante escoger en una etapa temprana del desarrollo la metodología de desarrollo a seguir. Por consiguiente, es importante evaluar las distintas opciones y escoger la que mejor se adapte al proyecto.

Hay dos corrientes de desarrollo: la tradicional y la ágil. La corriente tradicional es ventajosa cuando los requisitos están claros desde el principio, cosa que normalmente no ocurre, lo que produce un desarrollo más rígido y, por lo tanto, mayores costes y uso de recursos. En cambio, la metodología ágil es más flexible.

Al trabajar en equipo, no tener claro desde el principio los requisitos y tener la posibilidad de fragmentar las tareas, se decidió optar por usar una metodología de desarrollo ágil, en concreto la metodología SCRUM.

Al usar esta metodología se cumplen diversos puntos relacionados con la metodología SCRUM:

- **Versiones pequeñas.** Se realizan iteraciones cortas que duran entre una y dos semanas. Una vez terminada la iteración, se muestra al cliente lo desarrollado y se evalúa si es necesario hacer cambios en lo desarrollado.
- **Pruebas.** Después de cada iteración se realizan las pruebas necesarias.
- **Reuniones periódicas con los miembros del equipo.** Se realizan reuniones para revisar el estado del proyecto, organizar lo que se va a hacer tras una iteración y repartir tareas.
- **Integración.** Cada iteración será integrada en el producto final, ya sea en la aplicación web, en el sistema multiagente o en la aplicación de escritorio.

4.2. Requisitos de la aplicación

Para poder realizar la simulación debe de haber un medio de interacción entre el usuario y el sistema encargado de la simulación, por ello es preciso desarrollar una aplicación que permita esa interacción. La aplicación web y la de escritorio tendrán los mismos requisitos, ya que ambas deben de realizar las mismas funciones permitiendo al usuario usar la aplicación web y la de escritorio indistintamente.

En la toma de requisitos iniciales se identificó los actores que intervienen en el sistema. Posteriormente, en base a los objetivos, se procedió a identificar los requisitos y clasificarlos en requisitos funcionales y no funcionales.

4.2.1. Actores

Hay tres actores que realizan acciones en el sistema:

- **Usuario no autenticado.** Sólo podrá acceder a la aplicación.

- **Usuario.** Tiene uso total del sistema pudiendo crear simulaciones, visualizarlas, borrarlas, etc.

4.2.2. Requisitos funcionales

La siguiente tabla contiene los requisitos funcionales del usuario no autenticado:

Requisito	Descripción
Acceder a la aplicación	El usuario no autenticado podrá acceder a la aplicación introduciendo su nombre de usuario y contraseña.

Tabla 3 - Requisitos funcionales del usuario no autenticado

Por último, los requisitos funcionales del usuario:

Requisito	Descripción
Crear entornos	El usuario podrá crear entornos. Dichos entornos deben 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.
Consultar entornos	El usuario podrá consultar los entornos creados. Al seleccionar uno, se visualizará toda la información asociada a ese entorno.
Editar entornos	El usuario podrá editar un entorno creado si no forma parte de una simulación.

Eliminar entornos	El usuario podrá eliminar un entorno si no depende ninguna simulación de él.
Crear enfermedades	El usuario podrá crear enfermedades. Dichas enfermedades deben permitir elegir un nombre, la distancia de infección, el tiempo mínimo, medio y máximo 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, medio y mínimo de incubación.
Consultar enfermedades	El usuario podrá consultar un listado de las enfermedades creadas. Al seleccionar una, se visualizará toda la información asociada a esa enfermedad.
Editar enfermedades	El usuario podrá editar una enfermedad si no forma parte de una simulación.
Eliminar enfermedades	El usuario podrá eliminar una enfermedad si no depende ninguna simulación de ella.
Hacer simulaciones	El usuario podrá realizar una simulación. Para ello deberá poder elegir un entorno y una enfermedad de los creados previamente. También deberá poder insertar distintos valores tales como el nombre de la simulación, el número de infectados iniciales, el número de personas expuestas iniciales, el número de personas muertas iniciales, el nivel de aceptancia, que es la predisposición de

	los individuos de aceptar que están enfermos, un humano y el número de días que durará la simulación.
Consultar simulaciones	El usuario podrá ver un listado de todas las simulaciones realizadas por el usuario. Al seleccionar una simulación, se mostrará información acerca de ella.
Visualizar gráfica de la simulación	El usuario 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 simulación	El usuario podrá visualizar el desarrollo de la simulación con respecto al tiempo.
Eliminar simulaciones	El usuario podrá eliminar cualquiera de sus simulaciones.
Navegación	La aplicación web debe facilitar la navegación entre distintas secciones.

Tabla 4 - Requisitos funcionales del usuario

4.2.3. Requisitos no funcionales

Se identifican los siguientes para la aplicación:

Requisito	Descripción
Usabilidad	La aplicación debe ser lo más intuitiva posible, haciendo que sea cómoda y fácil de usar por parte de cualquier tipo de usuario.
Rendimiento	Se minimizará, en la medida de lo posible, el tiempo de simulación.

Mantenimiento	Las partes del código cuya comprensión sea más compleja deberán estar bien comentado. Otras características deseables que deberá tener el código es una buena estructuración con su correspondiente sangrado. Estas características facilitarán su mantenimiento.
Integración	La aplicación web debe poder integrarse en un servidor de aplicaciones tipo GlassFish.
Interoperabilidad	Las aplicaciones deben poder conectarse con 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, especialmente desde Mozilla Firefox, Google Chrome y Safari. La aplicación de escritorio debe ser multiplataforma.
Almacenamiento	Los datos de los usuarios, de las enfermedades, de los entornos y de las simulaciones deben ser almacenados en una base de datos.
Interfaz	La interfaz debe ser agradable e intuitiva.

Tabla 5 - Requisitos no funcionales

4.3. Casos de uso de la aplicación web

Una vez definido los actores y los requisitos, se procede a identificar los distintos casos de uso. En la Ilustración 1 se puede observar un diagrama que muestra los casos de uso.

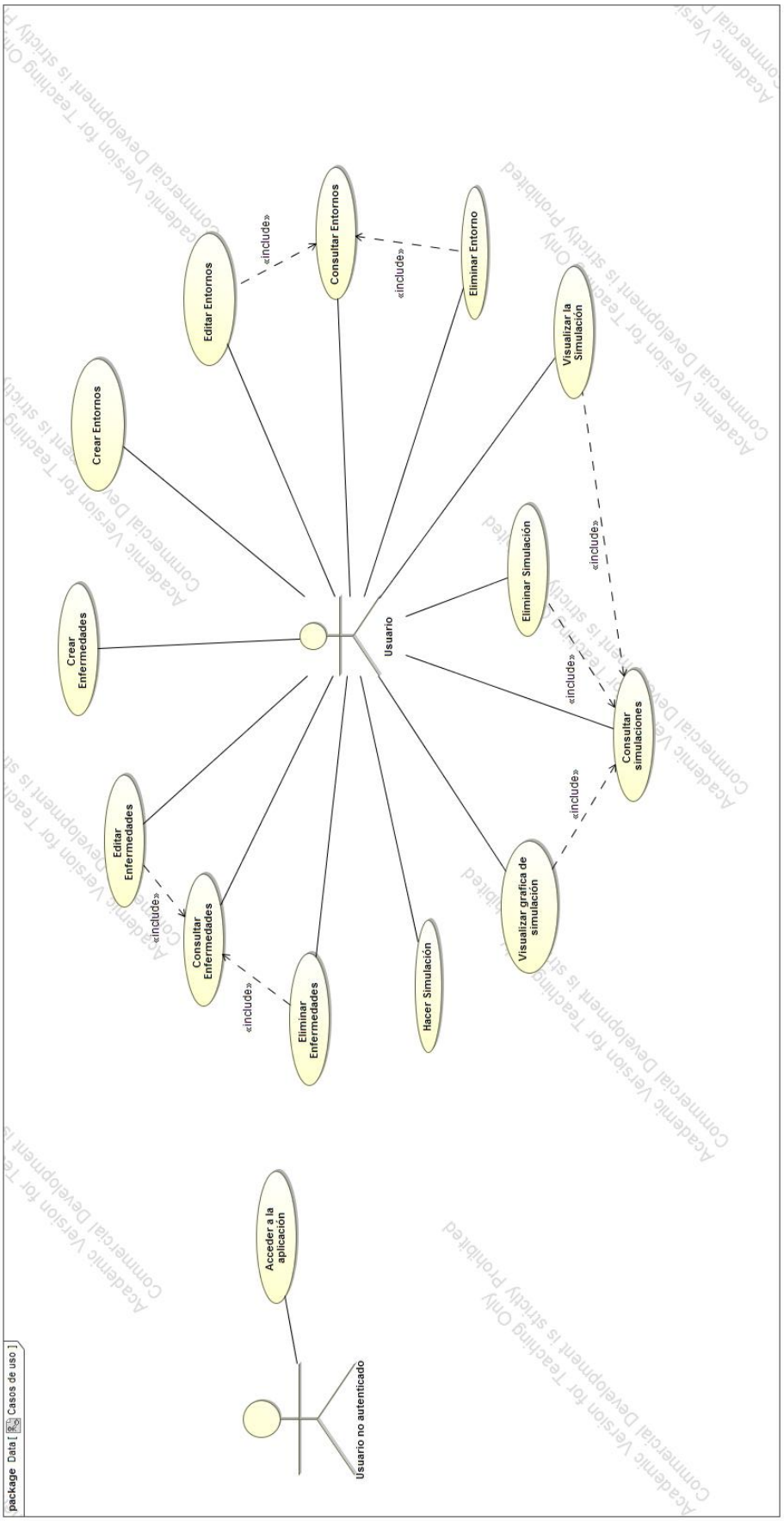

Ilustración 1 - Diagrama de casos de uso

A continuación se detalla los casos de usos:

4.3.1. Acceder a la aplicación

Descripción	El usuario no autenticado accede a la aplicación, se identifica y accede a la ventana principal.
Precondiciones	El usuario tiene que estar registrado.
Postcondiciones	El usuario está identificado y accede a la ventana principal.
Escenario principal	1. El usuario introduce el nombre de usuario. 2. El usuario introduce la contraseña. 3. Si el usuario y la contraseña son correctos, se muestra la ventana principal.
Escenario alternativo	1. El usuario introduce el nombre de usuario. 2. El usuario introduce la contraseña. 3. Si el usuario y la contraseña no son correctos, se muestra un mensaje de error indicando que los datos son incorrectos.
Información adicional	

Tabla 6 - CU Acceder a la aplicación

4.3.2. Crear entornos

Descripción	Un usuario crea un nuevo entorno.
Precondiciones	El usuario está identificado.
Postcondiciones	Se crea un nuevo entorno y lo muestra en la página de entornos.
Escenario principal	1. El usuario accede a la página de entornos. 2. El usuario completa todos los datos del formulario (todos son obligatorios): a. Nombre b. Número de habitantes

	c. Área d. Nivel de desarrollo e. Porcentaje de personas con mayor riesgo f. Media de personas por casa g. Media de personas por trabajo h. Porcentaje de personas que estudian i. Porcentaje de personas que trabajan 3. El usuario pulsa el botón crear. 4. Si todos los datos son correctos, 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 1,2 y 3 son iguales al escenario principal. 4. Si algún dato es incorrecto, se muestra un mensaje informando de los errores.
Información adicional	

Tabla 7 - CU Crear entornos

4.3.4. Consultar entornos

Descripción	El usuario consulta todos los entornos.
Precondiciones	El usuario está identificado.
Postcondiciones	Se muestra una lista de entornos creados por el usuario y se visualizan los datos del entorno seleccionado.
Escenario principal	1. El usuario accede a los entornos. 2. El usuario selecciona un entorno. 3. Se muestran los datos del entorno seleccionado.

Escenario secundario	1. El usuario accede a los entornos. 2. No hay ningún entorno que seleccionar, ya que la lista está vacía.
Información adicional	Se muestra una lista con los entornos creados por el usuario.

Tabla 8 - CU Consultar entornos

4.3.5. Editar entornos

Descripción	El usuario edita un entorno.
Precondiciones	El usuario está identificado y el entorno a editar no es utilizado en ninguna simulación.
Postcondiciones	Se guardan los nuevos valores del entorno.
Escenario principal	1. El usuario consulta los entornos (CU 4.3.4). 2. El usuario edita los campos. 3. El usuario pulsa el botón editar. 4. Si los campos son correctos, se guardan los nuevos datos en la base de datos.
Escenario secundario	1. El usuario consulta el entorno (CU 4.3.4). 2. El usuario edita los campos. 3. El usuario pulsa el botón editar. 4. Si los campos son incorrectos, se muestra un mensaje de error informando de los campos incorrectos.
Información adicional	

Tabla 9 - CU Editar entornos

4.3.6. Eliminar entornos

Descripción	El usuario elimina un entorno.
Precondiciones	El usuario está identificado y el entorno a eliminar no es utilizado en ninguna simulación.
Postcondiciones	El entorno seleccionado es eliminado
Escenario principal	1. El usuario consulta los entornos (CU 4.3.4). 2. El usuario pulsa el botón eliminar. 3. El entorno se elimina correctamente
Escenario secundario	1. El usuario consulta el entorno (CU 4.3.4). 2. El usuario pulsa el botón eliminar. 3. Como el entorno es utilizado en una simulación, se muestra el mensaje de error correspondiente.
Información adicional	

Tabla 10 - CU Eliminar entornos

4.3.7. Crear enfermedades

Descripción	Un usuario crea una nueva enfermedad.
Precondiciones	El usuario está identificado.
Postcondiciones	Se crea una nueva enfermedad que es mostrada en la página de enfermedades.
Escenario principal	1. El usuario accede a la página de enfermedades. 2. El usuario completa todos los datos del formulario (todos son obligatorios): a. Nombre b. Distancia de infección c. Tipo de enfermedad d. Tiempo medio de infección e. Tiempo máximo de infección

	f. Tiempo mínimo de infección g. Tiempo medio de incubación h. Tiempo máximo de incubación i. Tiempo mínimo de incubación j. Ratio de muerte k. Ratio de infectividad 3. El usuario pulsa el botón crear. 4. Si todos los datos son correctos, 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 1,2 y 3 son iguales al escenario principal. 4. Si algún dato es incorrecto, se muestra un mensaje informando de los errores.
Información adicional	Los campos g,h e i serán mostrados cuando se vaya a crear una enfermedad de tipo SEIS o SEIR.

Tabla 11 - CU Crear enfermedades

4.3.8. Consultar enfermedades

Descripción	El usuario consulta todas las enfermedades.
Precondiciones	El usuario está identificado.
Postcondiciones	Se muestra una lista de las enfermedades creadas por el usuario y se visualizan los datos de la enfermedad seleccionado.
Escenario principal	1. El usuario accede a las enfermedades. 2. El usuario selecciona una enfermedad. 3. Se muestran los datos de la enfermedad seleccionada.

Escenario secundario	1. El usuario accede a las enfermedades. 2. No hay ninguna enfermedad que seleccionar, ya que la lista está vacía.
Información adicional	Se muestra una lista con las enfermedades creadas por el usuario. Los campos relacionados con la incubación sólo se mostrarán si la enfermedad es de tipo SEIS o SEIR.

Tabla 12 - CU Consultar enfermedades

4.3.8. Editar enfermedades

Descripción	El usuario edita una enfermedad.
Precondiciones	El usuario está identificado y la enfermedad a editar no es utilizada en ninguna simulación.
Postcondiciones	Se guardan los nuevos valores de la enfermedad.
Escenario principal	1. El usuario consulta la enfermedad (CU 4.3.8). 2. El usuario edita los campos. 3. El usuario pulsa el botón editar. 4. Si los campos son correctos, se guardan los nuevos datos en la base de datos.
Escenario secundario	1. El usuario consulta la enfermedad (CU 4.3.8). 2. El usuario edita los campos. 3. El usuario pulsa el botón editar. 4. Si los campos son incorrectos, se muestra un mensaje de error informando de los campos incorrectos.
Información adicional	Los campos relacionados con la incubación sólo se mostrarán si la enfermedad es de tipo SEIS o SEIR.

Tabla 13 - Editar enfermedades

4.3.9. Eliminar enfermedades

Descripción	El usuario elimina una enfermedad.
Precondiciones	El usuario está identificado y la enfermedad a eliminar no es utilizada en ninguna simulación.
Postcondiciones	La enfermedad seleccionada es eliminada
Escenario principal	1. El usuario consulta la enfermedad (CU 4.3.8). 2. El usuario pulsa el botón eliminar. 3. La enfermedad se elimina correctamente.
Escenario secundario	1. El usuario consulta la enfermedad (CU 4.3.8). 2. El usuario pulsa el botón eliminar. 3. Como la enfermedad es utilizada en una simulación, se muestra el mensaje de error correspondiente.
Información adicional	

Tabla 14 - CU Eliminar enfermedades

4.3.10. Hacer simulación

Descripción	El usuario hace una simulación.
Precondiciones	El usuario está identificado. Debe de haber una enfermedad y un entorno creados.
Postcondiciones	La simulación es realizada y los datos de la simulación se almacenan.
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. El usuario completa todos los datos del formulario (todos son obligatorios): a. Nombre

	b. Número de días c. Número de infectados iniciales d. Número de personas en periodo de incubación iniciales e. Número de muertos iniciales f. Aceptancia 4. El usuario pulsa el botón simular. 5. Si todos los datos son correctos, comienza la simulación. 6. Al cabo de un tiempo, cuando la simulación haya terminado, se mostrará un mensaje notificando que la simulación ha terminado correctamente.
Escenario secundario	Los pasos 1,2 y 3 son iguales al escenario principal. 4. Si algún dato es incorrecto, se muestra un mensaje de error indicando cuáles son los parámetros incorrectos.
Información adicional	El campo d será mostrado si la enfermedad es de tipo SEIS o SEIR.

Tabla 15 - CU Hacer simulación

4.3.11. Consultar simulaciones

Descripción	El usuario consulta todas las simulaciones realizadas.
Precondiciones	El usuario está identificado.
Postcondiciones	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 accede a las simulaciones. 2. El usuario selecciona una simulación. 3. Se muestran los datos de la simulación seleccionada.

Escenario secundario	1. El usuario accede a las simulaciones. 2. No hay ninguna simulación que seleccionar, ya que la lista está vacía.
Información adicional	

Tabla 16 - CU Consultar simulaciones

4.3.12. Visualizar gráfica de la simulación

Descripción	El usuario puede visualizar en forma de gráfica algunos valores de simulación.
Precondiciones	El usuario está identificado y existe alguna simulación
Postcondiciones	Se muestra una gráfica con los valores de la simulación seleccionada.
Escenario principal	1. El usuario consulta una simulación (CU 4.3.11). 2. El usuario pulsa el botón ver gráfica. 3. Se muestran una gráfica de la simulación seleccionada.
Escenario secundario	1. El usuario consulta una simulación (CU 4.3.11). 2. No existe ninguna simulación, por lo que no se puede mostrar ninguna gráfica.
Información adicional	

Tabla 17 - CU Visualizar gráfica de la simulación

4.3.13. 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.

Precondiciones	El usuario está identificado y existe alguna simulación
Postcondiciones	Se muestra gráficamente la simulación.
Escenario principal	1. El usuario consulta una simulación (CU 4.3.11). 2. El usuario pulsa el botón ver simulación. 3. Se muestran la simulación seleccionada.
Escenario secundario	1. El usuario consulta una simulación (CU 4.3.11). 2. No existe ninguna simulación, por lo que no se puede mostrar ninguna simulación.
Información adicional	

Tabla 18 - CU Visualizar simulación

4.3.14. Eliminar simulación

Descripción	El usuario elimina una simulación.
Precondiciones	El usuario está identificado y existe alguna simulación.
Postcondiciones	La simulación seleccionada es eliminada.
Escenario principal	4. El usuario consulta la simulación (CU 4.3.11). 5. El usuario pulsa el botón eliminar. 6. La simulación se elimina correctamente.
Escenario secundario	
Información adicional	

Tabla 19 - CU Eliminar simulación

4.4. Requisitos de la simulación

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. A continuación se lista 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 estos lugares frecuentes 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 no se puede contagiar a otros humanos. Al final de la incubación, el humano enferma.

Muerte	El humano puede morir a causa de una enfermedad.

Tabla 20 - Requisitos funcionales de la simulación

4.5. Análisis de la base de datos

La base de datos contendrá 5 entidades:

- **disease.** Almacenará las enfermedades creadas.
- **environment.** Almacenará los entornos creados.
- **graphicsData.** Almacenará la información de la simulación mostrada en la gráfica.
- **humanInstant.** Almacenará la información de cada humano en cada instante de la simulación.
- **level_development.** Almacenará los distintos niveles de desarrollo que puede tener un entorno.
- **simulation.** Almacenará las simulaciones realizadas.
- **typedisease.** Almacenará los distintos tipos de enfermedad que puede tener una enfermedad.
- **user.** Almacenará los usuarios de la aplicación.

4.6. Análisis de la Interfaz de Usuario

Los siguientes puntos identifican lo que se debe aplicar a la interfaz de usuario de la aplicación web:

- **Aspecto gráfico.** Para dar mayor impacto visual se ha aplicado CSS y Primefaces. El uso de Primefaces permite ventajas como ahorro en tiempo de desarrollo y una alta compatibilidad con los componentes JSF.
- **Navegación.** Se ofrecerá al usuario un menú en la parte superior izquierda que le permite acceder a las principales secciones de la aplicación web.

4.7. Limitaciones y restricciones

El sistema tiene ciertas restricciones:

- La aplicación web debe poder integrarse en un servidor de aplicaciones GlassFish.
- El código fuente del proyecto y sus comentarios estará escrito en inglés.
- El idioma de las aplicaciones será el inglés.

5. Diseño

5.1. Diagrama de distribución

En la Ilustración 2 se muestran los componentes del sistema a alto nivel.

Ilustración 2 - Diagrama de distribución

El lado del cliente está formado por el navegador Web, que se comunica de manera bidireccional con la aplicación a través del servidor de aplicaciones. La comunicación se realiza mediante llamada HTTP, estas llamadas también incluyen las llamadas asíncronas AJAX. Otro componente del lado del cliente es la aplicación de escritorio, esta aplicación se conecta con el servidor a través de servicios web RESTful.

El servidor de aplicaciones, que contiene la aplicación web y los servicios RESTful, se comunica con el servidor de bases de datos y con JADE, y éstos con el servidor de aplicaciones. JADE también se comunica bidireccionalmente con el servidor de base de datos.

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

5.2. Modelo relacional de la base de datos

Toda la información del sistema es guardada en una base de datos, para la gestión de toda esta información se ha utilizado el modelo relacional de la Ilustración 3:

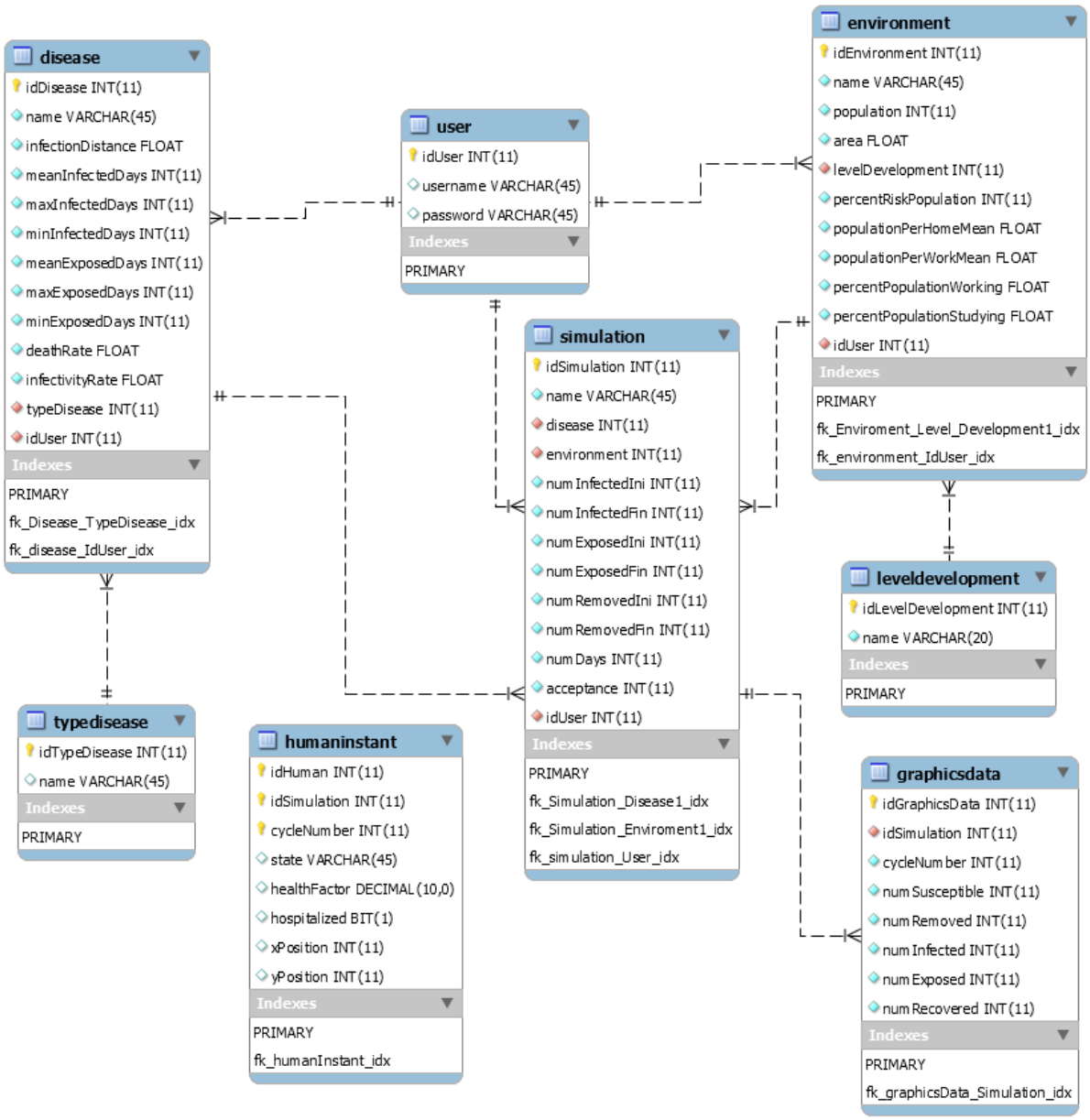

Ilustración 3 - Diagrama de base de datos

A continuación se muestra la descripción detallada de cada tabla:

- **disease.** Almacena las enfermedades creadas por el usuario. La comprobación de que los datos están bien introducidos se hace desde las aplicaciones.
  - **idDisease.** Clave primaria que identifica la enfermedad, asignada por el índice *PRIMARY*.
  - **name.** Nombre de la enfermedad.
  - **infectionDistance.** Distancia máxima a la que puede contagiar la enfermedad.
  - **meanInfectedDays.** Media de días que dura la enfermedad una vez infectado el humano.

- **maxInfectedDays.** Número máximo de días que dura la enfermedad una vez infectado el humano.
- **minInfectedDays.** Número mínimo de días que dura la enfermedad una vez infectado el humano.
- **meanExposedDays.** Media de días que dura la incubación una vez expuesto el humano.
- **maxExposedDays.** Número máximo de días que dura la incubación una vez expuesto el humano.
- **minExposedDays.** Número mínimo de días que dura la incubación una vez expuesto el humano.
- **deathRate.** Tasa de mortalidad de la enfermedad.
- **infectivityRate.** Tasa de infectividad de la enfermedad.
- **typeDisease.** Tipo de enfermedad. Este campo es una clave foránea a la clave primaria de la tabla typedisease, asignada por el índice fk_Disease_TypeDisease_idx.
- **idUser.** Usuario que creó la enfermedad. Este campo es una clave foránea a la clave primaria de la tabla user, asignada por el índice fk_Disease_idUser_idx.

– **typeDisease.** Almacena los tipos de enfermedades existentes.
  - **idTypeDisease.** Clave primaria que identifica al tipo de enfermedad, asignada por el índice *PRIMARY*.
  - **name.** Nombre del tipo de enfermedad.

– **environment.** Almacena los entornos creados por los usuarios. La comprobación de que los datos están bien introducidos se hace desde las aplicaciones.
  - **idEnvironment.** Clave primaria que identifica el entorno, asignada por el índice *PRIMARY*.
  - **name.** Nombre del entorno.
  - **population.** Número de habitantes.
  - **area.** Área del entorno.
  - **levelDevelopment.** Nivel de desarrollo del entorno. Este campo es una clave foránea a la clave primaria de la tabla leveldevelopment, asignada por el índice fk_Environment_Level_Development1_idx.
  - **percentRiskPopulation.** Porcentaje de la población de riesgo.
  - **populationPerHomeMean.** Media de habitantes en casa.
  - **populationPerWorkMean.** Media de habitantes en el trabajo.
  - **percentPopulationWorking.** Porcentaje de personas trabajando.
  - **pecentPopulationStudying.** Porcentaje de personas estudiando.
  - **idUser.** Usuario que creó el entorno. Este campo es una clave foránea a la clave primaria de la tabla user, asignada por el índice fk_Environment_idUser_idx.

- **leveldevelopment.** Almacena los niveles de desarrollos existentes.
  - **idLevelDevelopment.** Clave primaria que identifica el nivel de desarrollo, asignada por el índice *PRIMARY*.
  - **name.** Nombre del nivel de desarrollo.

- **user.** Almacena los usuarios del sistema.
  - **idUser.** Clave primaria que identifica al usuario, asignada por el índice *PRIMARY*.
  - **username.** Nombre de usuario.
  - **password.** Contraseña del usuario.

- **simulation.** Almacena las simulaciones creadas. La comprobación de que los datos están bien introducidos se hace desde las aplicaciones.
  - **idSimulation.** Clave primaria que identifica a la simulación, asignada por el índice *PRIMARY*.
  - **name.** Nombre de la simulación.
  - **disease.** Enfermedad simulada. Este campo es una clave foránea a la clave primaria de la tabla disease, asignada por el índice fk_Simulation_Disease1_idx.
  - **environment.** Entorno simulado. Este campo es una clave foránea a la clave primaria de la tabla environment, asignada por el índice fk_Simulation_Environment1_idx.
  - **numInfectedIni.** Número de personas infectadas al principio de la simulación.
  - **numInfectedFin.** Número de personas infectadas al final de la simulación.
  - **numExposedIni.** Número de personas expuestas al principio de la simulación.
  - **numExposedFin.** Número de personas expuestas al final de la simulación.
  - **numRemovedIni.** Número de personas muertas al principio de la simulación.
  - **numRemovedFin**. Número de personas muertas al final de la simulación.
  - **numDays.** Número de días que dura la simulación.
  - **acceptance.** Aceptancia de la población de que están enfermos.
  - **idUser.** Usuario que creó el entorno. Este campo es una clave foránea a la clave primaria de la tabla user, asignada por el índice fk_simulation_idUser_idx.

- **graphicsdata.** Almacena los datos que aparecerán en la gráfica de la simulación.
    - **idGraphicsData.** Clave primaria que identifica a los datos de la gráfica, asignada por el índice *PRIMARY*.
    - **idSimulation.** Simulación a la que representa la gráfica. Este campo es una clave foránea a la clave primaria de la tabla simulation, asignada por el índice fk_graphicsData_Simulation_idx.
    - **cycleNumber.** Instante de la simulación.
    - **numSusceptible.** Número de personas sanas en ese instante.
    - **numRemoved.** Número de muertos en ese instante.
    - **numInfected.** Número de infectados en ese instante.
    - **numRecovered.** Número de personas recuperadas en ese instante.

- **humanInstant.** Almacena la información de las personas en cada instante.
    - **idHuman.** Clave primaria que identifica al humano al que pertenece la información, asignada por el índice *PRIMARY*.
    - **idSimulation.** Clave primaria que identifica la simulación a la que pertenece la información del humano, asignada por el índice *PRIMARY*.
    - **cycleNumber.** Clave primaria que identifica el instante de la simulación, asignada por el índice *PRIMARY*.
    - **state.** Estado de salud del humano.
    - **healthFactor.** Salud del humano de manera cuantitativa.
    - **hospitalized.** Informa si el humano está hospitalizado o no.
    - **xPosition.** Posición x del humano en el entorno.
    - **yPosition.** Posición y del humano en el entorno.

5.3. Patrones de diseño

Con el fin de evitar fallos posibles en las aplicaciones y que el resultado final sea de calidad, se ha usado el patrón de diseño **Singleton**. Este patrón se aplica a la hora de iniciar simulaciones y evita, mientras se está realizando una simulación, iniciar otra simulación si el usuario pulsa el botón.

5.4. Diagrama de clases

A continuación se muestran los diagramas con las clases que componen las clases del proyecto. Las clases del proyecto se han dividido en tres diagramas: un diagrama de clases que representa el sistema multiagente, un diagrama de clases que representa la aplicación web y un diagrama de clases que representa la aplicación de escritorio.

5.4.1. Diagrama de clases del sistema multiagente

En la Ilustración 4 se muestra el diagrama de clases correspondiente al sistema multiagente.

- Las clases Living Being, MainController y SecondaryController son clases extendidas de la clase Agent, que es una clase de JADE. A su vez, Human extiende de LivingBeing que representa a los humanos del entorno. Cada uno de los agentes tienen asociados un comportamiento.
- Disease, es creada por el MainController y representa a la enfermedad. También está asociada con Human y Environment, esto se debe a que un humano enferma con esa enfermedad y que está presente en un entorno.
- Human, representa a un humano que vive en un entorno. Un humano se encuentra en una posición (clase Position) y tiene asociado diferentes posiciones, como son la casa, el trabajo (si trabaja) y los lugares que frecuenta.
- La clase Simulation controla la hora de la simulación.
- HumanStatus y SimulationStatus representan el estado de un humano en un instante y el estado de la simulación en un instante respectivamente.
- La clase Parameters guarda los datos de la simulación y utiliza la clase AgentGateway para comunicarse con el servidor. Para que no haya dos simulaciones a la vez se ha incluido el patrón de diseño Singleton representado por la clase Singleton.

Ilustración 4 - Diagrama de clases del sistema multiagente

5.4.2. Diagrama de clases de la aplicación web

Debido a la gran cantidad de clases que tiene la aplicación web se ha dividido en dos además, se ha simplificado el diagrama de clases incluido en este documento, adjuntándose el diagrama de clases completo en el recurso electrónico. En la ilustración 5 se muestra el grueso de la aplicación web.

- En la primera fila se encuentra la clase AbstractFacade. Esta clase abstracta incluye los métodos esenciales para acceder a los datos de la base de datos.
- En la segunda fila se encuentra las Facade de las entidades de la base de datos. Estas Facade implementa los métodos necesarios para acceder a los datos de la entidad.
- En la tercera fila se encuentran las interfaces de las Facade anteriormente descritas.
- En la cuarta fila se encuentran los Managed Beans, estas clases son el modelo para las vistas de la aplicación y también contiene la lógica de la aplicación web.
- Por último se encuentran las entidades de la base de datos. Estas clases representan la persistencia de la base de datos.

En la Ilustración 6 se muestra los servicios RESTful y la clase de comunicación entre el sistema multiagente y la aplicación (AgentGateway) con sus correspondientes dependencias.

- AbstractFacade es similar a la clase con el mismo nombre anteriormente descrita. La única diferencia es que ésta está programada para ser utilizada como servicio RESTful.
- Lo mismo ocurre con las demás Facade de este diagrama.
- La clase AgentGateway representa la clase que comunica la aplicación web con el sistema multiagente. De ella depende Parameters que son los datos de la simulación y Singleton.

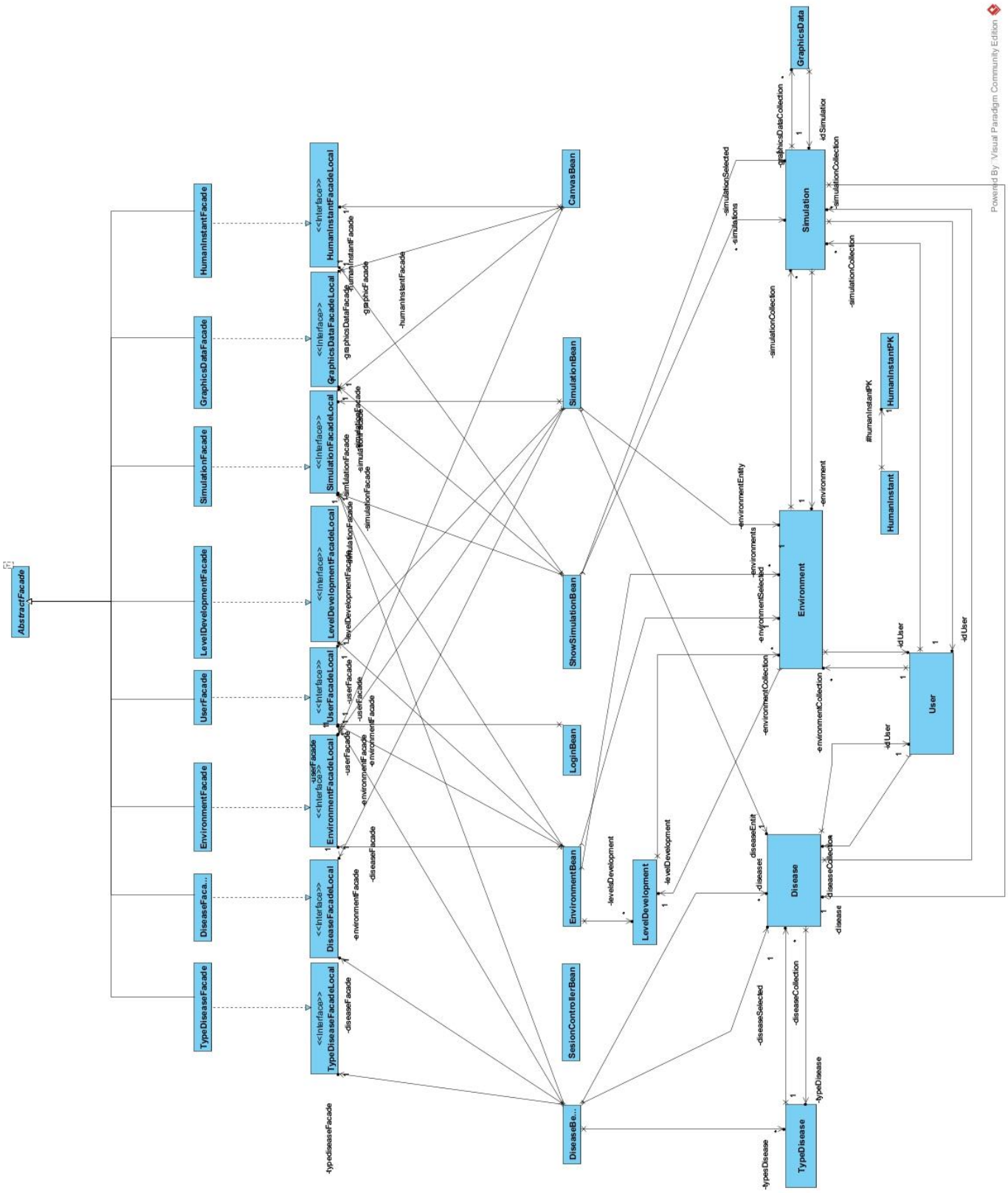

Ilustración 5 - Diagrama de clases de la aplicación web 1

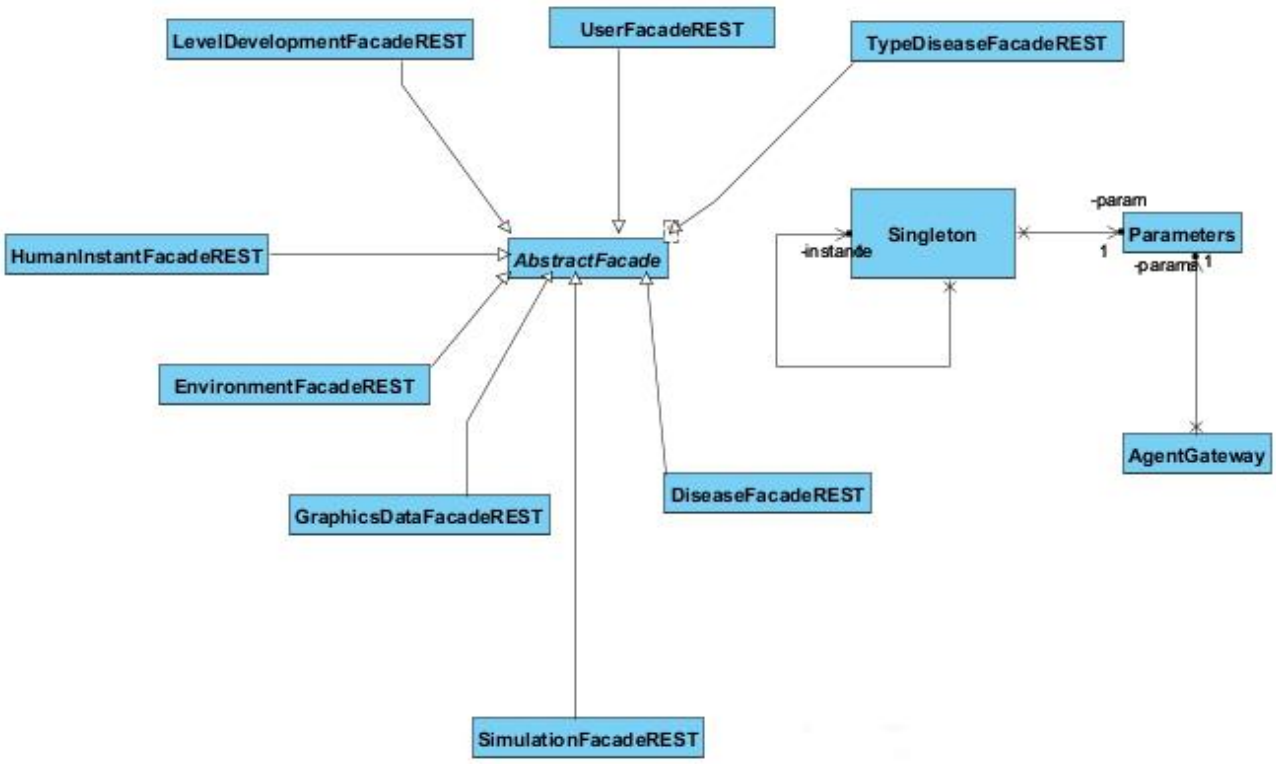

Ilustración 6 - Diagrama de clases de la aplicación web 2

5.4.3. Diagrama de clases de la aplicación de escritorio

Al igual que ocurre con los diagramas de clase de la aplicación web, los diagramas de clases de la aplicación de escritorio incluido en este documento es uno simplificado, adjuntando el completo en el recurso electrónico. El diagrama de clases de la aplicación de escritorio está ilustrado en la Ilustración 7. La aplicación de escritorio sigue el patrón MVC (Modelo-Vista-Controlador).

- Las clases modelo son las entidades de la base de datos. Estas clases son Disease, TypeDisease, User, Environment, LevelDevelopment, Simulation, HumanInstant, HumanInstantPK y GraphicsData.
- La clase controlador es Controller.
- Las clases de la vista son SimulationView, ShowSimulationView, LoginView y ChartSimulation.
- Las clases WorkerDoSimulation, WorkerDeleteSimulation y WorkerShowSimulation permiten la concurrencia de la aplicación y evita posibles errores que puedan ocurrir debido a la concurrencia.

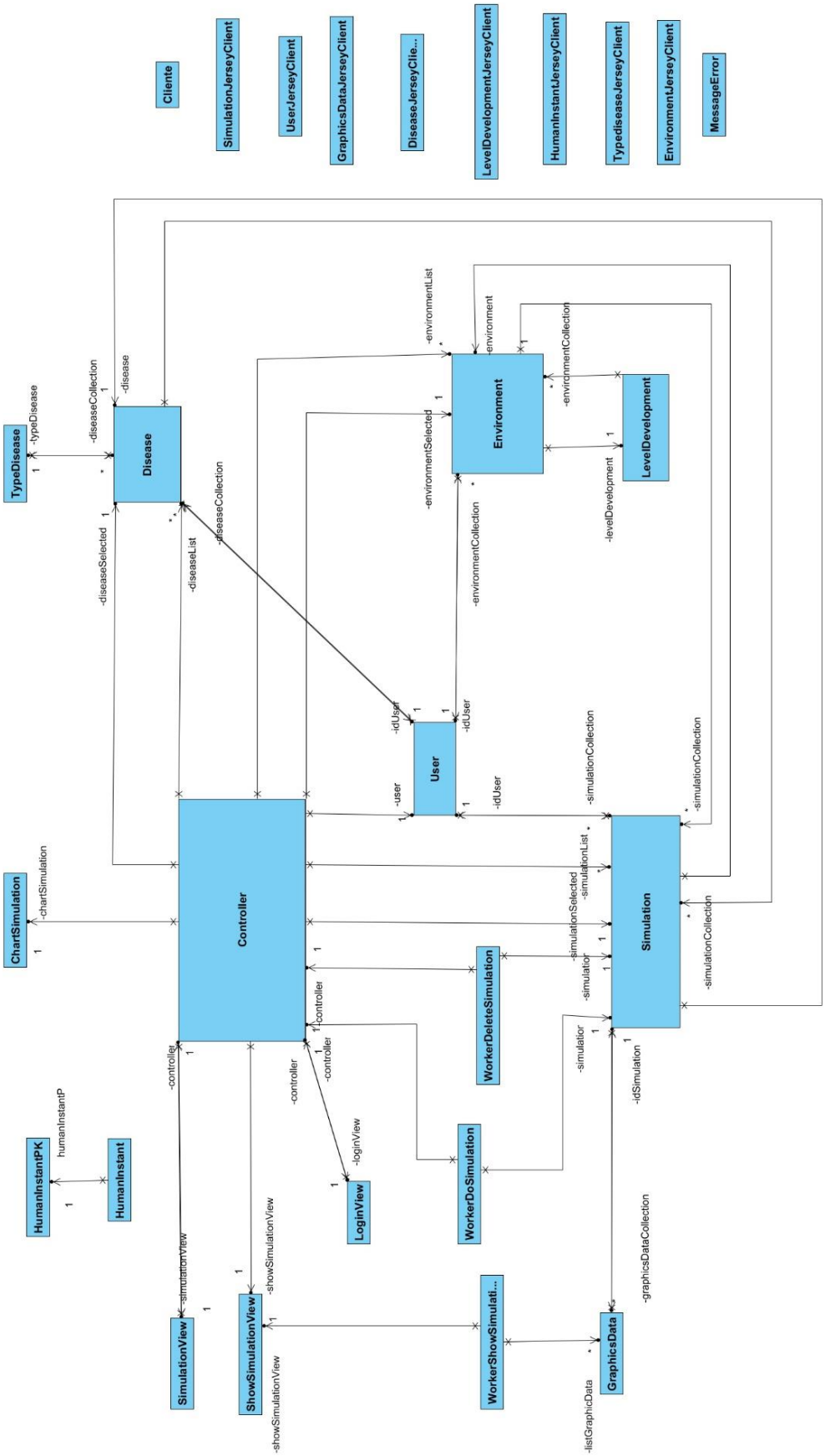

Ilustración 7 - Diagrama de clases de la aplicación de escritorio

- Las clases SimulationJerseyClient, UserJerseyClient, GraphicsDataJerseyClient, DiseaseJerseyClient, LevelDevelopmentJerseyClient, HumanInstantJerseyClient, TypeDiseaseJerseyClient y EnvironmentJerseyClient acceden a los datos de la base de datos.
- MessageError es una clase que se encarga de mostrar un mensaje cuando hay un error.
- Client es la clase principal de la aplicación.

5.5. Modelo de los agentes

Para que la aplicación fuera óptima y cumpliera todas las especificaciones había que hacer un diseño de los agentes. La simulación sólo requiere de un agente, el humano, pero como el sistema no es independiente, ya que las simulaciones son solicitadas a través de un usuario en el lado del cliente, era necesario implementar un controlador que pusiera en marcha la simulación en el caso de que recibiese una petición por parte del usuario. En un diseño inicial, sólo existían este controlador y los humanos pero, al realizar pruebas a la aplicación se comprobó que se sobrecargaba mucho el sistema provocando pérdidas de mensajes y un funcionamiento poco óptimo.

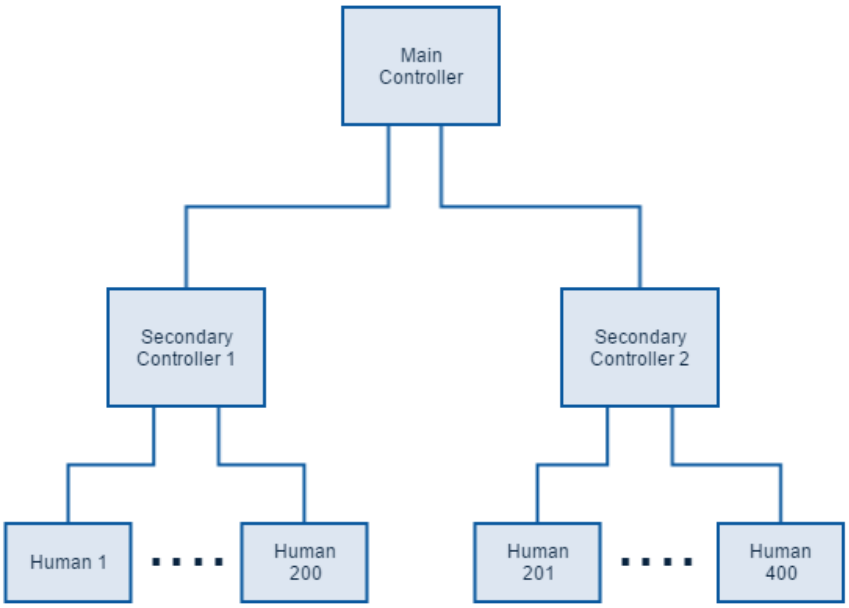

Ilustración 8 - Diseño final de los agentes

Esto hizo que se planteara un nuevo modelo de agentes. Para ello se ha implementado una especie de maestro-esclavo, donde hay un controlador principal que recibe la petición y que inicia la simulación creando un número de controladores secundarios en función del número de habitantes del entorno. Estos controladores secundarios realizan la función del antiguo controlador y, una vez acabada la

simulación, envían toda la información de los humanos al controlador principal que se encarga de ensamblarla. Esta mejora supuso que no se perdiesen mensajes y que el sistema funcionara muchísimo mejor.

6. Implementación y pruebas

6.1. Estructura del proyecto

A continuación se ilustra la estructura del proyecto de la aplicación web y la estructura del proyecto de la aplicación de escritorio, explicando brevemente cada una de sus partes.

6.1.1. Estructura del proyecto de la aplicación web

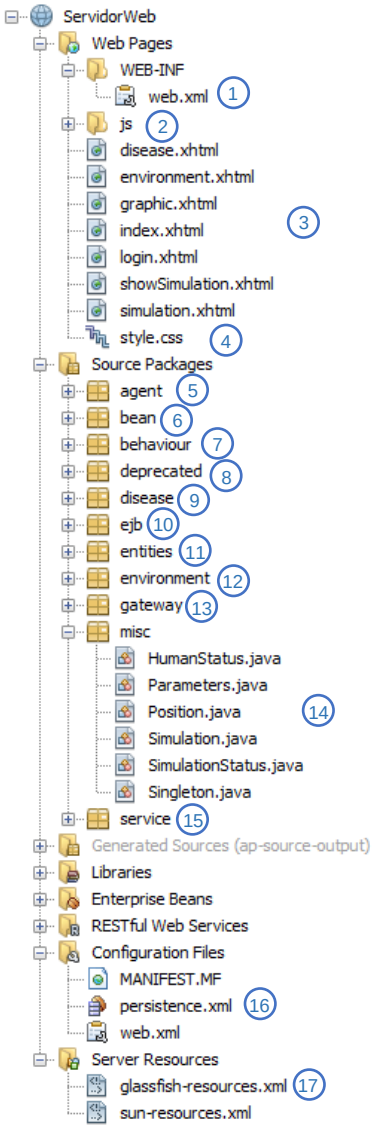

Ilustración 9 - Estructura del proyecto de la aplicación web

1. Archivo de configuración de las páginas.
2. Carpeta que contiene todos los archivos JavaScript.
3. Vistas de la aplicación web.
4. Archivo CSS de la aplicación.
5. Paquete que contiene los agentes del sistema multiagente.
6. Paquete que contiene los Managed Beans de la aplicación web.
7. Paquete que contiene los comportamientos de los agentes.
8. Paquete que contiene los archivos de versiones antiguas.

9. Paquete que contiene la clase Disease.
10. Paquete que contiene los Facade que permite la comunicación con la base de datos.
11. Paquete que contiene las entidades de la base de datos.
12. Paquete que contiene la clase Environment.
13. Paquete que contiene AgentGateway que permite la comunicación con el sistema multiagente.
14. Paquete que contiene clases auxiliares.
15. Paquete que contiene los servicios RESTful.
16. Archivo de configuración de la persistencia.
17. Archivo de configuración del servidor.

6.1.2. Estructura del proyecto de la aplicación de escritorio

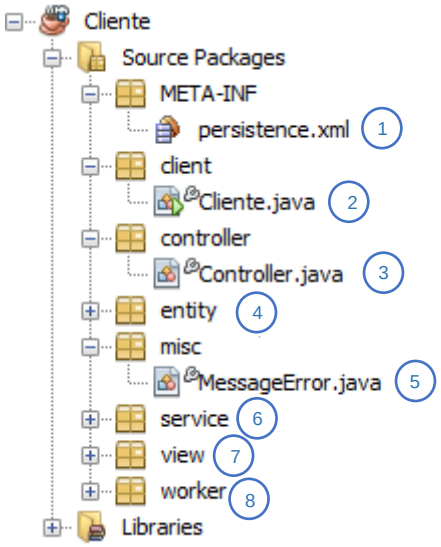

Ilustración 10 - Estructura del proyecto de la aplicación de escritorio

1. Archivo de configuración de la persistencia.
2. Clase principal de la aplicación.
3. Clase que se encarga de controlar la aplicación.
4. Paquete que contiene las entidades de la base de datos.
5. Clase que se encarga de mostrar un mensaje cuando hay un error.
6. Paquete que contiene los servicios RESTful que permiten la comunicación con la base de datos.
7. Paquete que contiene las vistas de la aplicación.
8. Paquete que contiene las clases que permiten la concurrencia y que no haya errores en la aplicación (workers).

6.2. Tareas implementadas

Al ser un trabajo desarrollado en equipo, se han dividido las diferentes tareas del proyecto entre todos. En mi caso, se ha desarrollado las siguientes tareas:

- Aplicación Web.
    - CRUD Environment.
    - Ver simulación.
    - Validar campos.
    - Desarrollo de interfaz gráfica y CSS.
- Aplicación escritorio.
    - CRUD Environment.
    - Validar campos.
    - Desarrollo de la interfaz gráfica.
- Sistema multiagente.
    - Human
        - Movimiento día laborable.
        - Lugares frecuentes.
        - Cambios de estado de salud.
        - Comportamiento.
    - Separación de los comportamientos de los agentes.
- Base de datos.
    - Desarrollo inicial de la base de datos.

De estas tareas hay ciertas tareas que, debido a su interesante funcionalidad dentro del sistema o su complejidad, es necesario explicar con más detalles.

6.2.1. Ver simulación

Para visualizar la simulación en la aplicación web se ha desarrollado un canvas. Este canvas es una cuadrícula que representa las distintas posiciones del entorno. Las posiciones donde hay humanos están coloreados de distinto color dependiendo del estado de los humanos. El color verde representa a los humanos sanos, el rojo a los infectados, el azul a los recuperados y el negro a los muertos. El canvas está implementado en el fichero JavaScript “Board.js”, en este fichero se dibuja la cuadrícula y las posiciones y estado de los humanos. La información de los humanos en cada instante es obtenida, en cada paso de la simulación, de la base de datos. Esto se debe a que cada instante de la información puede contener muchos datos, y si se trae de golpe toda la información al principio de la simulación no funciona de manera óptima la simulación, aparte de que Java no puede soportar tal magnitud de datos en un solo instante.

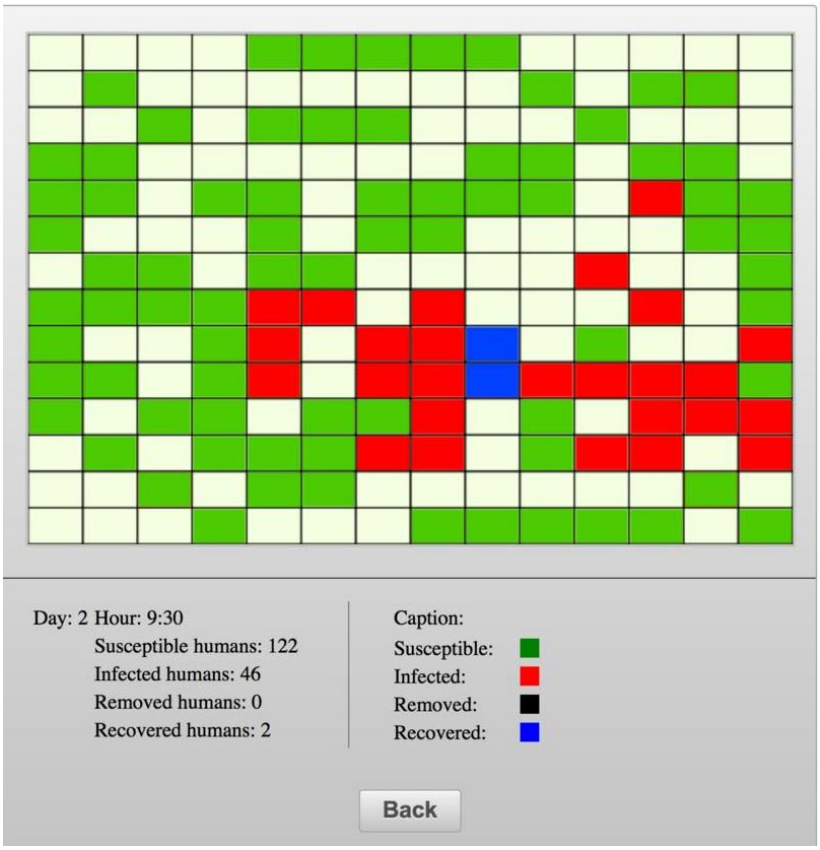

Ilustración 11 - Visualización de una simulación

Además de la cuadrícula se muestran datos informativos tales como la hora, el día y el número de personas sanas, infectadas, recuperadas y muertas en ese instante. Tanto la cuadrícula como la información se actualizan cada segundo.

El uso de AJAX es esencial en este caso porque la cuadrícula se tiene que actualizar en cada instante sin tener que actualizar la vista. El hecho de utilizar JSF permite utilizar el framework Primefaces que implementa algunas etiquetas para utilizar AJAX. Para comunicar el código JavaScript con el bean que gestiona la vista se utiliza la etiqueta remoteCommand, esta etiqueta permite ejecutar métodos del bean desde JavaScript, en este caso ejecuta un método en el bean que accede a la base de datos y obtiene toda la información relativa a ese instante. Esa información es enviada a JavaScript en formato JSON posibilitando que JavaScript pueda guardar esa información. Para actualizar los datos informativos también se hace uso de Primefaces. Para ello se utiliza la etiqueta poll que envía peticiones AJAX de manera periódica, estas peticiones ejecutan un método en el bean que incrementa la hora. El número de muertos, sanos, infectados y recuperados son obtenidos a la vez que la información de los humanos en ese instante.

6.2.2. Movimiento día laborable

De 00:00 de la noche a 8:00 de la mañana, se asume que el humano duerme. De 8:00 a 16:00 va a trabajar. Una vez sale del trabajo, tiene la tarde libre así que toma

aleatoriamente una decisión de a dónde ir. Está decisión produce tres posibles situaciones:

- **Ir a casa.** Puede que tras salir del trabajo el humano esté cansado del trabajo. Debido a esto hay mayor probabilidad de que vaya a casa a descansar, en concreto, un 50% de posibilidad.
- **Lugar frecuente.** Puede dirigirse también a algún lugar frecuente. Cada lugar frecuente de la lista tiene un 10% de probabilidad.
- **Movimiento libre.** Moverse libremente a través del entorno tiene un 10% de probabilidad.

En un tarde puede ir a más de un sitio, debido a que el tiempo que pasa en un sitio es variable. A las 23:00, se dirige a casa.

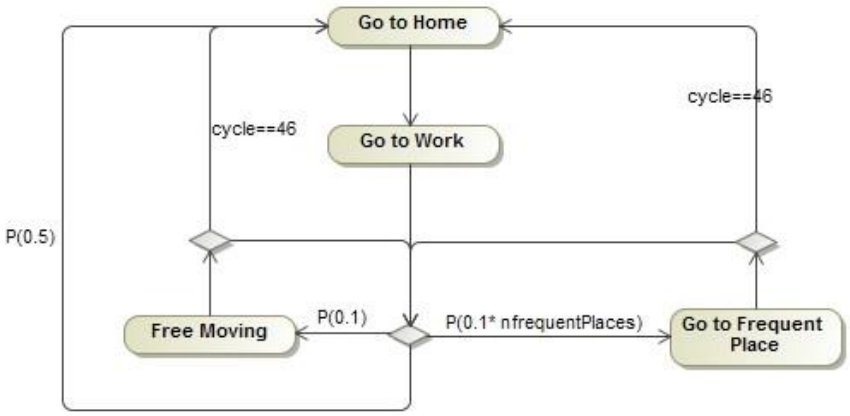

Ilustración 12 - Diagrama de actividad del movimiento del humano en día laborable

6.2.3. Lugares frecuentes

El humano tiene dos lugares frecuentes desde el inicio de la simulación, su casa y el trabajo si trabaja. A parte de estos lugares frecuentes, tiene una lista de lugares frecuentes a los que suele ir. Esta lista es dinámica, es decir, cada vez que un humano se está moviendo aleatoriamente comprueba si el sitio al que ha llegado lo ha visitado anteriormente. Si es así, se aumenta el contador de visitas a ese sitio y si no, se crea una entrada en un mapa con ese sitio como clave y un 1 como valor. Una vez haya visitado un sitio cinco veces se añade a la lista de sitios frecuentes.

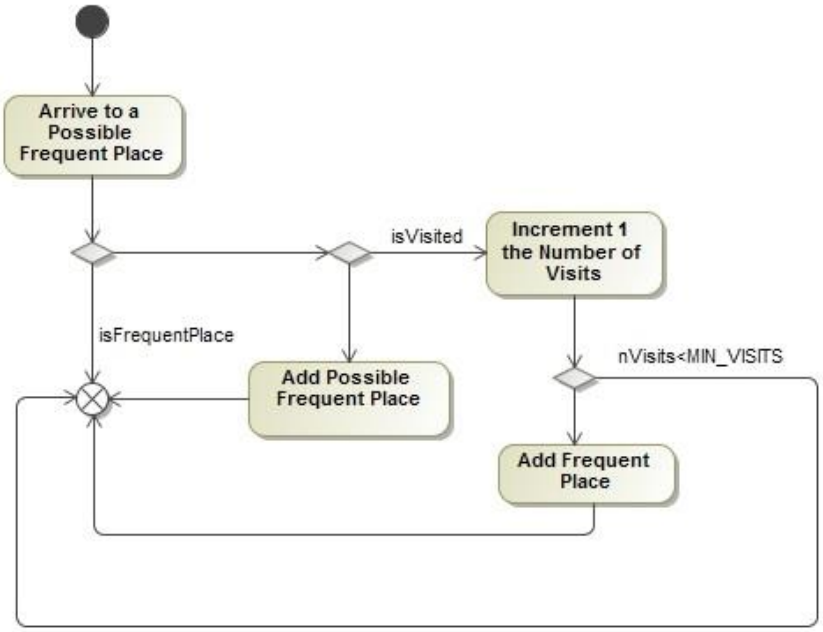

Ilustración 13 - Diagrama de actividad de lugares frecuentes

6.2.4. Cambios de estado de salud

El estado de salud se modifica cuando empieza un nuevo día, excepto cuando se contagia la enfermedad que puede ser en cualquier momento del día. Si el humano está incubando la enfermedad y el número de días que dura la incubación acaba, el humano se infecta. En el caso de que esté infectado, si el número de días que dura la infección acaba, entonces el humano puede morir o si la enfermedad es de tipo SIR o SEIR, recuperarse y ser inmune a la enfermedad o recuperarse pudiendo contagiarse de nuevo.

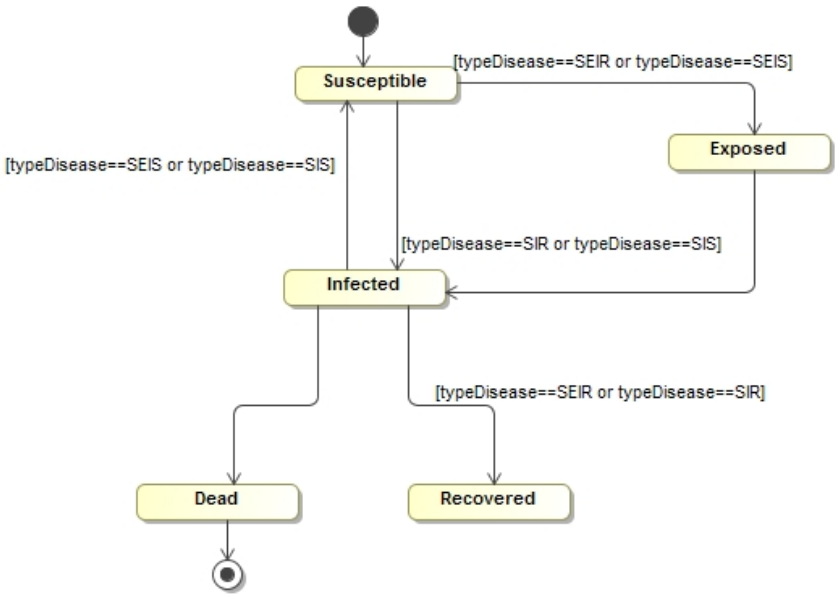

Ilustración 14 - Diagrama de estados de salud

6.2.5. Comportamiento del humano

Un humano tiene dos maneras de comportarse, depende de si trabaja o no. El humano trabajador comprueba si ese día trabaja (hay un 74.5% de posibilidad de ir a trabajar), si trabaja y está enfermo decide si ir a trabajar o quedarse en casa. En el caso de que vaya a trabajar se comportará como si fuese un día laborable (véase apartado 6.2.2), en cambio si decide no ir a trabajar o no tiene que trabajar ese día se comportará como si fuese un día no laborable. Los que no trabajan siempre se comportarán como si fuese un día laborable y no harán la comprobación inicial que hacen los que trabajan.

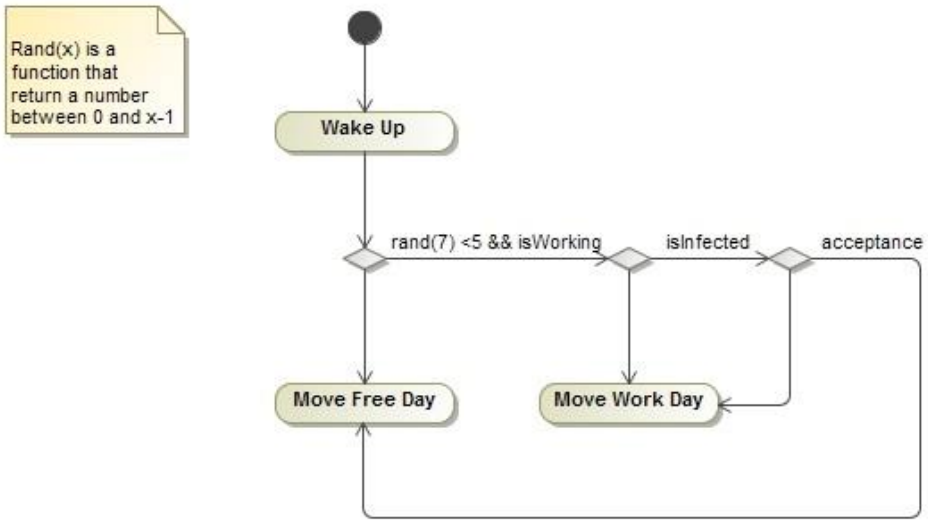

Ilustración 15 - Diagrama de actividad del comportamiento del humano

6.2.6. Separación de los comportamientos de los agentes.

Para tener un código de mejor calidad y más legible, se separó el humano en dos partes, modelo y controlador. El modelo es la clase Human donde se implementan los métodos que definen el objeto y el controlador es la clase HumanBehaviour que controla el comportamiento del humano utilizando los métodos del modelo.

6.3. Método de trabajo

Para realizar la implementación se ha seguido una serie de metodologías:

- **Notación estándar Java.** Se utiliza la notación CamelCase para los identificadores de las clases y la notación dromedaryCase para la definición de métodos y variables.
- **Formateado e indentación.** El código se ha formateado e indentado durante toda la implementación.

- **Manejo de errores excepciones.** Se tratan los errores que puedan producirse y se maneja de forma adecuada.

6.4. Pruebas

Al seguirse una metodología SCRUM, cada vez que se termina una iteración se ha realizado pruebas para comprobar que todo funcionase correctamente. Estas pruebas no fueron automatizadas, ya que, debido a la magnitud del proyecto, no se disponía de las horas necesarias para realizar pruebas automatizadas como por ejemplo pruebas unitarias. No obstante, se han realizado ciertas pruebas de estrés y también se ha medido lo que tarda ciertas tareas para comprobar el rendimiento del sistema multiagente.

Las pruebas se han realizado sobre ordenadores portátiles por lo que éstas están limitadas, debido a que este sistema requiere mucho procesamiento y mucha memoria al tratarse un gran volumen de datos. En unos equipos más potentes, como unos servidores dedicados, podría limitarse menos. Debido a esto y la incapacidad de poder utilizar unos equipos más potentes, se ha limitado la aplicación en función de las pruebas realizadas en un equipo 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.** 6 GB
- **SO.** Windows 10 Home. 64bits.

Los resultados de las pruebas son los siguientes:

Número de agentes (hebras)	Número de días	Cantidad de memoria virtual (MB)	Funciona	Comentario
2000	10	512	Sí	Funcionamiento muy lento.
2000	30	512	No	La simulación se tuvo que parar debido al lento funcionamiento.
1200	30	512	Sí	Pérdida de datos de un agente por desbordamiento de memoria.
500	60	512	Sí	
750	60	512	No	Fallo de desbordamiento de memoria.
750	60	1024	Sí	

1300	30	1024	Sí	
2000	30	2048	Sí	Mejora mucho el rendimiento con respecto a la primera prueba con 2000 agentes.
2000	60	2048	No	Desbordamiento de memoria.
2000	50	2048	Sí	
1500	70	2048	Sí	
1500	80	2048	No	Desbordamiento de memoria.
1000	110	2048	Sí	
1000	120	2048	Sí	
1000	130	2048	Sí	
1000	140	2048	No	Pérdida de mensajes por desbordamiento de memoria.
500	150	2048	Sí	
500	170	2048	Sí	
500	200	2048	No	Pérdida de mensajes por desbordamiento de memoria.
500	185	2048	No	Pérdida de mensajes por desbordamiento de memoria.

Tabla 21 - Pruebas realizadas

Se ha limitado el número máximo de días que varían en función del volumen de habitantes del entorno a simular. En base a estos resultados, se han establecido los siguientes máximos:

- Entre 0 y 500 habitantes, el máximo de días es 170.
- Entre 500 y 1000 habitantes, el máximo de días es 130.
- Entre 1000 y 1500 habitantes, el máximo de días es 70.
- Entre 1500 y 2000 habitantes, el máximo de días 50.

Como se puede observar, el rendimiento mejora conforme se aumenta la memoria RAM de la máquina virtual de Java. También se realizaron dos mejoras con el fin de obtener un mejor rendimiento. La primera fue la jerarquía de controladores comentada en la Sección 5.5, y la segunda fue la inserción, al final de la simulación, por parte de los controladores secundarios de toda la información de sus humanos, en vez del controlador principal.

7. Conclusiones y mejoras futuras

7.1. Conclusiones

Este proyecto permite a los usuarios disponer de una herramienta para poder simular enfermedades en distintos entornos, cumpliendo con los objetivos que en un principio se propusieron. La herramienta desarrollada en este proyecto es una herramienta básica que abre la posibilidad de ampliarse en un futuro. Estas ampliaciones y mejoras pueden realizarse por la comunidad debido al espíritu Open Source que cuenta el proyecto desde un principio. Esto permitirá tener una amplia diversidad de funcionalidades, mayor soporte y supondrá ahorro en costes de desarrollo y mantenimiento de los mismos. A parte, el proyecto se ha realizado de manera que cada usuario pueda desarrollar su propio cliente, puesto que puede obtener toda la información a partir de la base de datos a través de consultas SQL.

Personalmente, este proyecto ha sido muy satisfactorio y he sacado provecho de ello. El tener que trabajar con nuevas tecnologías como son los sistemas multiagente o trabajar en equipo utilizando control de versiones y tomando decisiones cada semana me ha aportado experiencia y madurez en ciertos aspectos relacionados con el desarrollo.

7.2. Futuras líneas de trabajo

Dado que la realización de un TFG no cubre todas las horas necesarias para poder realizar una versión con todas las funcionalidades posibles, y que este proyecto puede llegar a ser inmenso, se ha recopilado una serie de posibles mejoras o líneas de trabajo que puede tener este proyecto:

- **Inclusión de grupos de usuarios.** Esto permite que un usuario pertenezca a un grupo de usuarios que compartan las mismas simulaciones, entornos y enfermedades. Esto sería útil en laboratorios de investigación por ejemplo.
- **Modelar los datos.** A partir de una serie de datos modelarlos de manera que las fórmulas para contagiar, calcular número de días de infección o de exposición se adapten a los datos.
- **Centralizar base de datos.** Centralizar la información generada en distintos clientes permite a los usuarios ver las simulaciones de otros usuarios y así tener mayor conocimiento.
- **Mejorar movimiento de los humanos.** Actualmente, los humanos se mueven de un sitio a otro, a excepción del movimiento libre, de manera directa. Con esta mejora, se puede implementar que un humano se mueva de posición en posición siguiendo una heurística como puede ser la distancia Manhattan.
- **Simulación con varias enfermedades.** Un entorno con varias enfermedades puede dar mucho juego y a la vez ser más realista.
- **Inclusión de medios de transporte.** Se pueden incluir medios de transporte en el que se pueda realizar contagios. Por ejemplo, no es lo mismo ir al trabajo en coche que ir en un autobús donde hay más posibilidades de contagio.

- **Inclusión de animales.** Se pueden incluir animales que pueden ser vectores de contagio. Estos animales pueden ser domésticos o salvajes.
- **Simulación de más de un entorno.** JADE permite distribuir los agentes en diferentes computadoras. Distribuir la aplicación puede permitir que cada computador simule un entorno. Además, al poder moverse los agentes entre un computador u otro, se puede implementar la funcionalidad de que un humano pueda viajar de un entorno a otro asemejándose más a la realidad.
- **Hospitalización de enfermos.** Un enfermo puede ser hospitalizado. La hospitalización puede disminuir el número de días infectados y la probabilidad de muerte.
- **Vacunas.** Inclusión de vacunas frente a la enfermedad.

Como se puede observar, hay multitud de posibles mejoras siendo éstas sólo algunas de ellas.

Bibliografía

Página Oficial de JADE: http://jade.tilab.com/

Página Oficial de Mason: https://cs.gmu.edu/~eclab/projects/mason/

Página Oficial de MadKit: http://www.madkit.org/

Página Oficial de Repast: http://repast.sourceforge.net/

Página Oficial de Netlogo: https://ccl.northwestern.edu/netlogo/

Wikipedia: http://wikipedia.org/

StackOverflow: http://stackoverflow.com

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

PrimeFaces 5.2 User Guide:
http://www.primefaces.org/docs/guide/primefaces_user_guide_5_2.pdf

JADE 4.3.3 API: http://jade.tilab.com/doc/api/

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

Gson 2.3.1 API:

http://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/index.html

Jill Bigley Dunham. (2005). *An Agent-Based Spatially Explicit Epidemiological Model in MASON.* Journal of Artificial Societies and Social Simulation vol. 9, no. 1.

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

Ana Mas (2005). *Agentes Software y Sistemas Multiagete: Conceptos, Arquitecturas y Aplicaciones.* Madrid: Prentice Hall.

Apéndices

A. Manual de instalación

- **Requisitos mínimos**

Para poder ejecutar el sistema la máquina virtual de Java tiene que tener como mínimo 2GB de RAM.

- **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/. 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 puede ser cualquiera.

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 es un 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, el IDE usado 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.

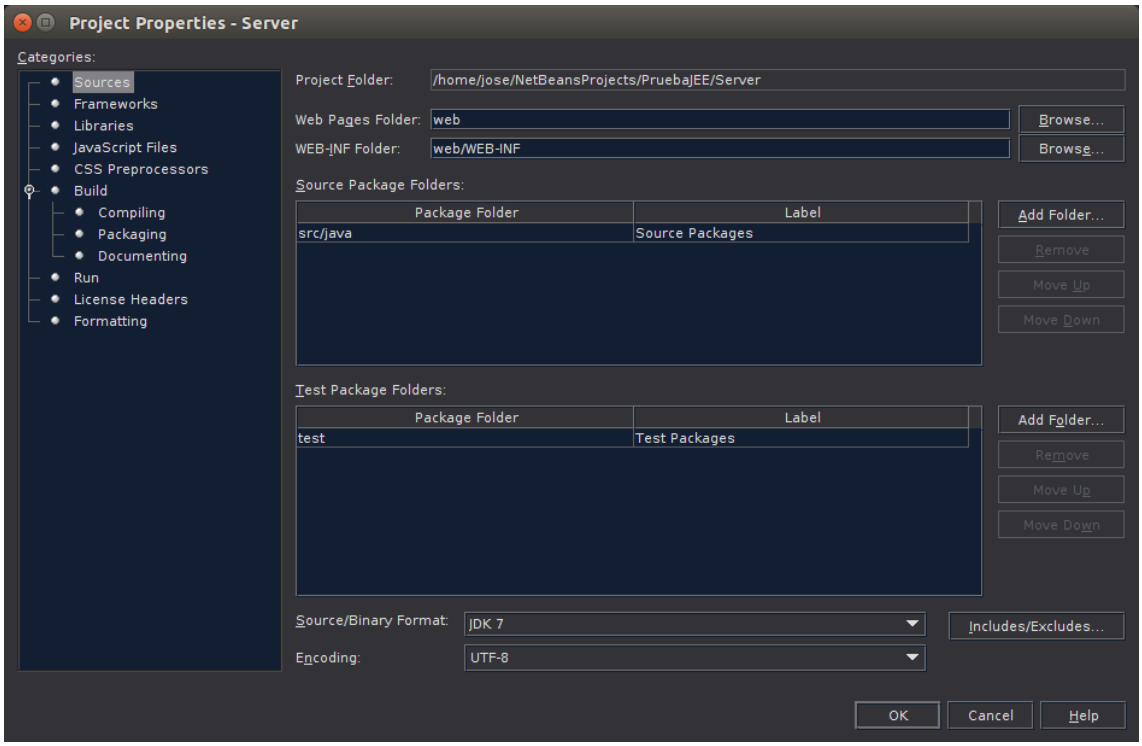

Ilustración 16 - Propiedades del proyecto

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.

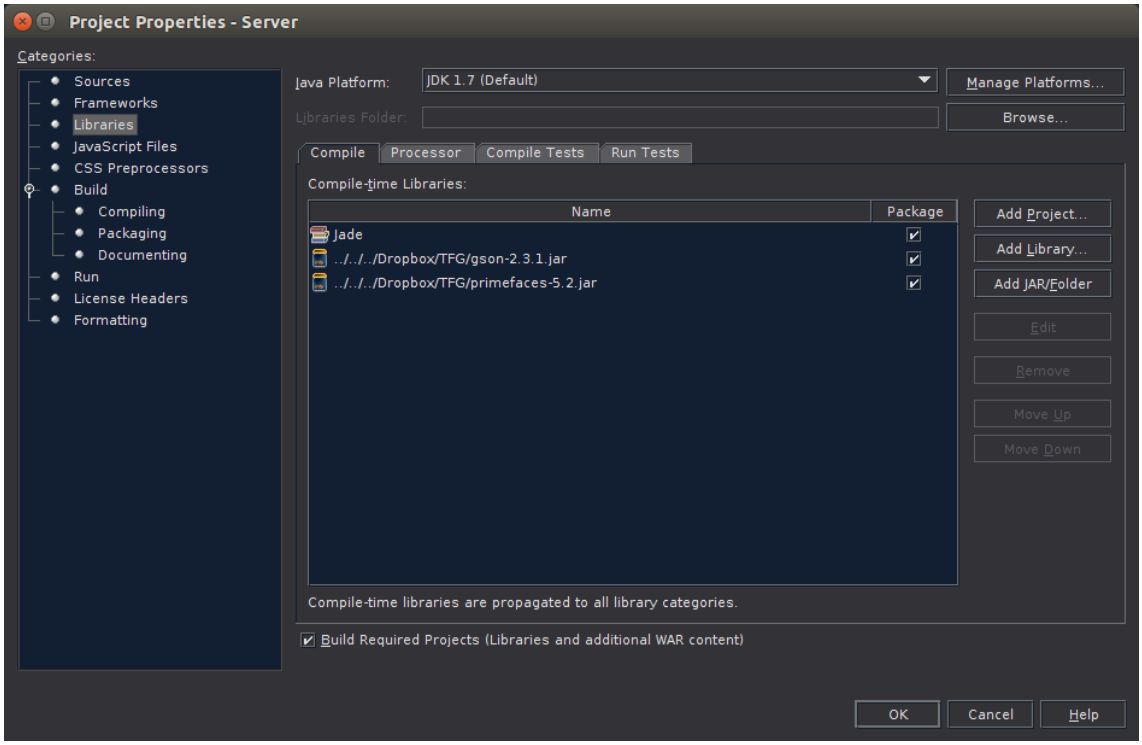

Ilustración 17 - Librerías que contiene el proyecto

- 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.

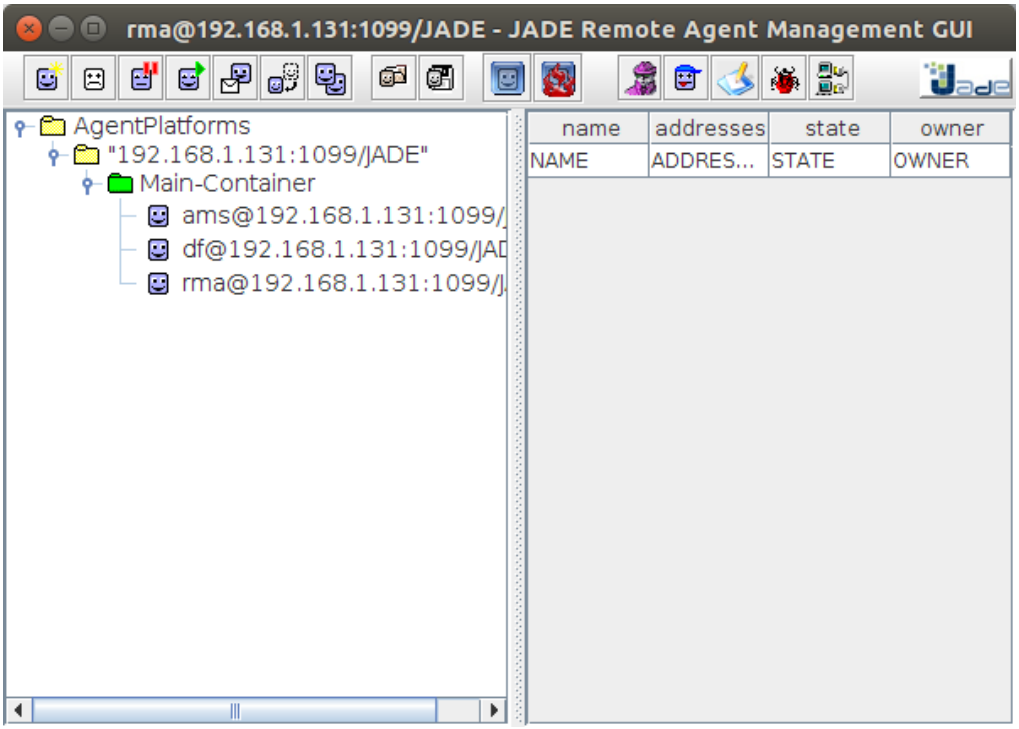

Ilustración 18 - Interfaz gráfica de JADE

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. Como la máquina virtual de Java tiene que tener mínimo 2GB de RAM, hace falta configurarla para que tenga esta capacidad, ya que por defecto tiene asignada 512MB. Esta configuración se realiza en el servidor GlassFish tal y como se detalla a continuación:

  – Configuración memoria RAM de la máquina virtual de Java

Hay que acceder a la consola de GlassFish, una vez abierta hay que desplegar, en el menú de la izquierda, “Configurations”. Posteriormente se clica sobre “JVM Settings” y, por último, selecciona la pestaña “JVM Options”. Una vez abierta, hay que realizar lo siguiente:

- Modificar el valor “-Xmx512” por el valor “-Xmx2048”.
- Cliclar sobre “Add JVM Option” y a añadir el siguiente valor: “-XXUseGCOverheadLimit”. Este valor desactiva el error “OutOfMemoryError” que se produce cuando el recolector de basura está en uso durante el 98% del tiempo de ejecución, y se recupera menos del 2% del montículo.

Por último, se pulsa el botón Save para guardar los cambios realizados.

- **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' en nuestro caso 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

a. Manual de usuario de la aplicación web

La página principal de la aplicación web es un formulario de ingreso a la aplicación. 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 19 - Vista de autenticación

Ilustración 20 - Autenticación fallida

La vista principal de la aplicación, una vez el usuario se ha autenticado, es la creación de simulaciones.

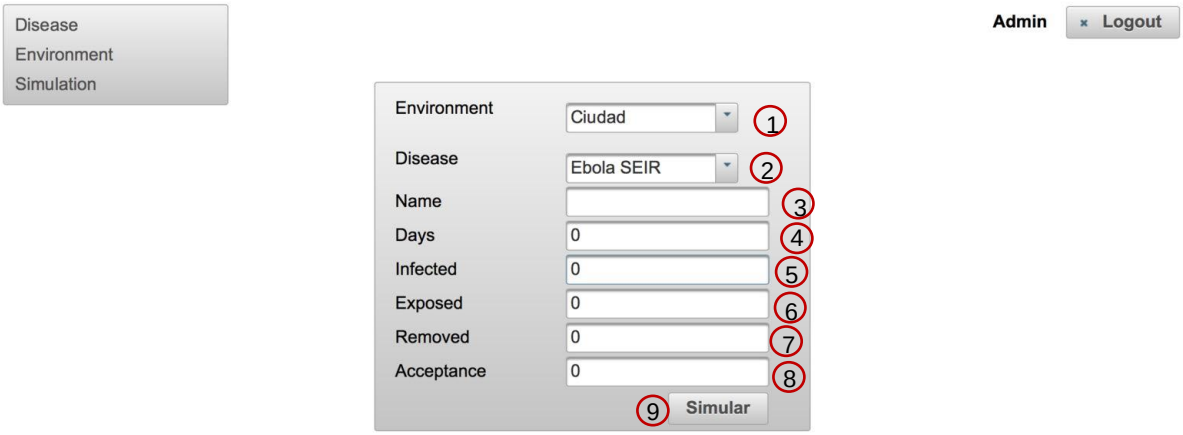

Ilustración 21 - Página principal, donde se crean las simulaciones, de la aplicación

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 nombre de una simulación ya existente. Además, el nombre tiene que tener como mínimo un carácter y como máximo 15.
4. Número de días que dura la simulación. Este campo debe ser un número entero entre 1 y el máximo de días que se pueden simular. El número máximo de días que se pueden simular se establece en función del número de habitantes del entorno seleccionado. Estos máximos son los siguientes:
    a. Entre 0 y 500 humanos. 170 días
    b. Entre 500 y 1000 humanos. 130 días
    c. Entre 1000 y 1500 humanos. 70 días
    d. Entre 1500 y 2000 humanos. 50 días
5. El mínimo de días es 1, y el máximo se establece en función del número de habitantes del entorno seleccionado. Estos máximos son los siguientes:
    a. Entre 0 y 500 humanos. 170 días
    b. Entre 500 y 1000 humanos. 130 días
    c. Entre 1000 y 1500 humanos. 70 días
    d. Entre 1500 y 2000 humanos. 50 días
6. Número de infectados iniciales. Este campo debe ser un número entre 0 y la población del entorno que se va a simular.
7. 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.

8. 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.
9. Aceptancia. Predisposición de los individuos de aceptar que están enfermos.
10. Botón de simulación. Al pulsarlo 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.

La página Disease permite crear, visualizar, editar y borrar una enfermedad.

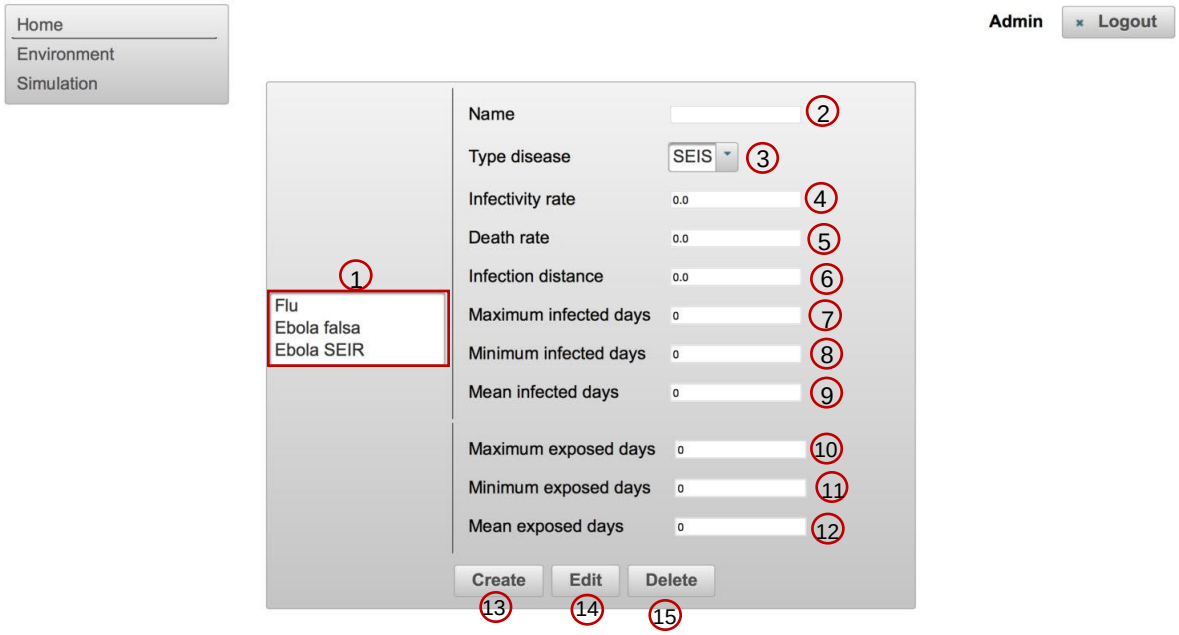

Ilustración 22 - Página Disease

1. Lista de enfermedades creadas.
2. Nombre de la enfermedad. El nombre no puede ser el mismo que el nombre de una ya existente. Además, el nombre tiene que tener como mínimo un carácter y como máximo 15.
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 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 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 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 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 mínimo 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.
13. Botón para crear enfermedades. Al pulsarlo se creará una enfermedad. Si el nombre de la enfermedad que se intenta crear ya existe, se mostrará un mensaje indicándolo.
14. Botón para editar una enfermedad. Al pulsarlo se editará una enfermedad. Si el nuevo nombre existe o hay una simulación relacionada con ella, se mostrará un mensaje indicando que no se puede editar.
15. Botón para borrar una enfermedad. Al pulsarlo se borrará. Si la enfermedad está relacionada con alguna simulación, se mostrará un mensaje indicando que no se puede borrar.

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

Ilustración 23 - Página Environment

1. Lista de los entornos creados.
2. Nombre del entorno. El nombre no puede ser el mismo que el 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 número máximo de habitantes puede ser definido por el administrador del sistema, por defecto es 1000 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 30 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 pulsarlo se creará un entorno. Si el nombre del entorno que se intenta crear ya existe, se mostrará un mensaje indicándolo.

12. Botón para editar un entorno. Al pulsarlo se editará un entorno. Si el nuevo nombre del entorno existe o hay una simulación relacionada con él, se mostrará un mensaje indicando que no se puede editar.
13. Botón para borrar un entorno. Al pulsarlo se borrará el entorno. Si el entorno está relacionado con alguna simulación, se mostrará un mensaje indicando que no se puede borrar.

La página Simulation muestra las simulaciones creadas y la información de cada una de ellas.

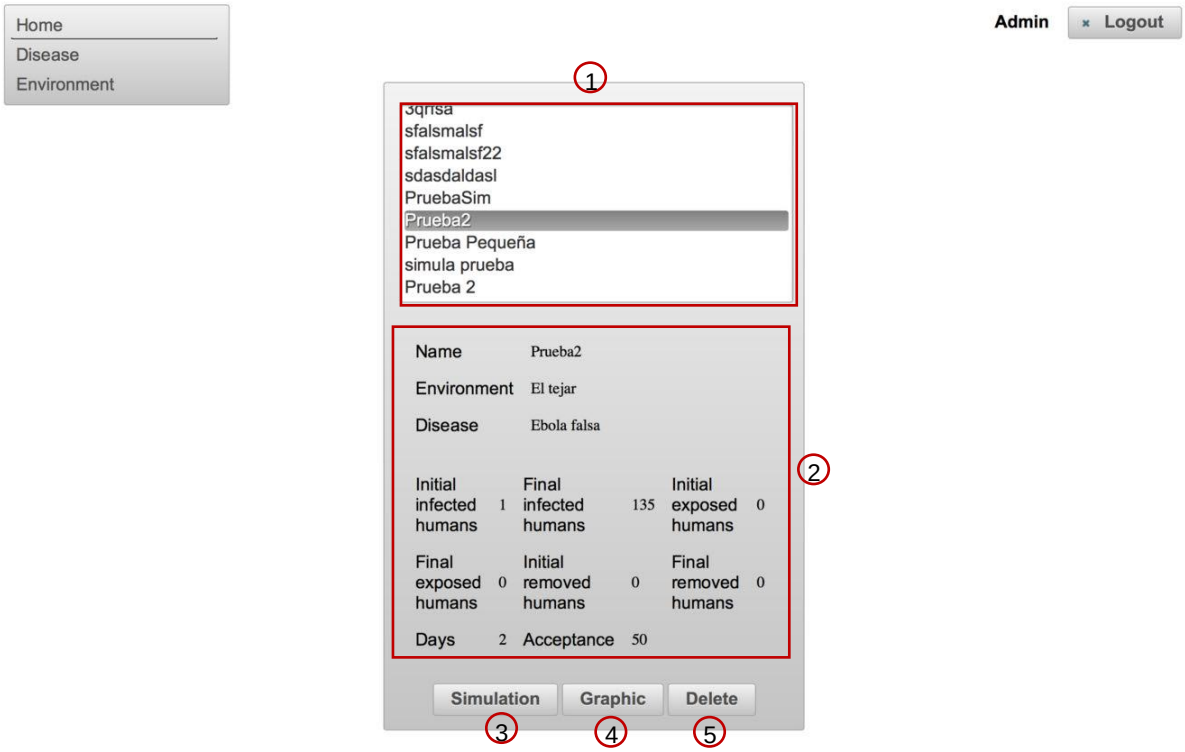

Ilustración 24 - Página Simulation

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 la vista Simulation si se pulsa el botón “Simulation” se accederá a la siguiente vista.

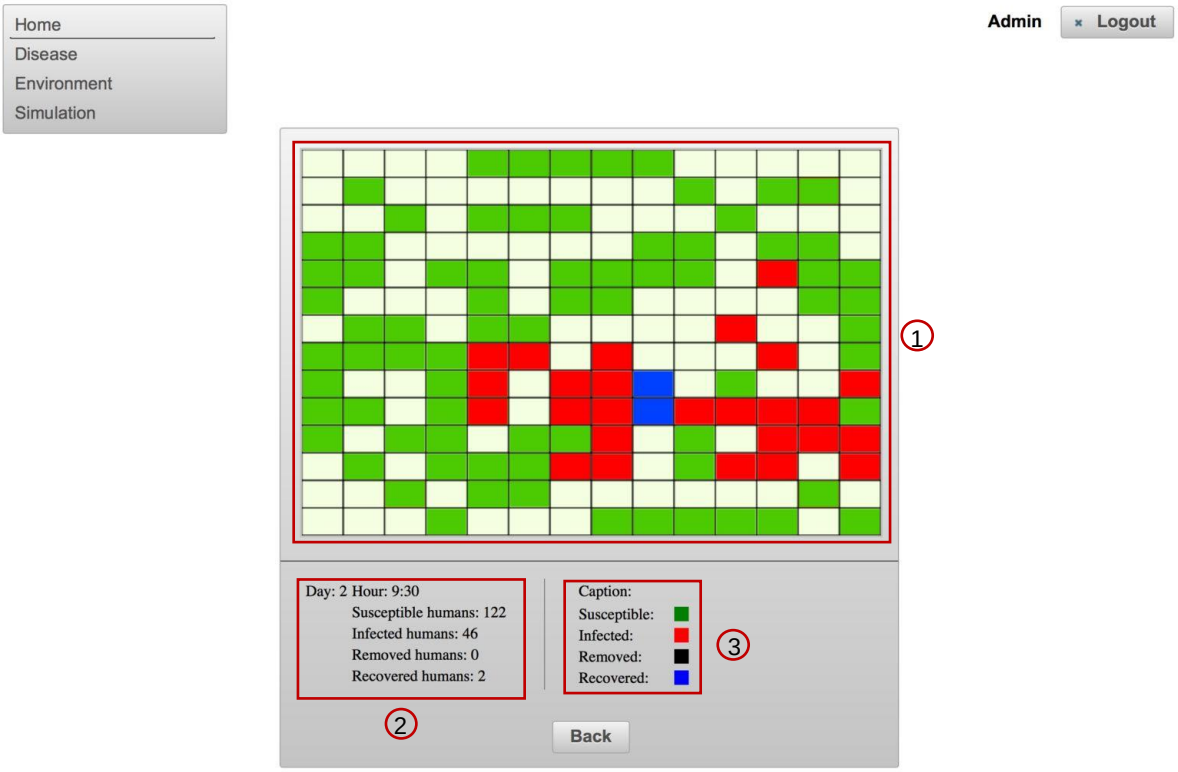

Ilustración 25 - Vista gráfica de la simulación

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.

En la página Simulation si se pulsa el botón “Graphic” se accederá a la siguiente vista.

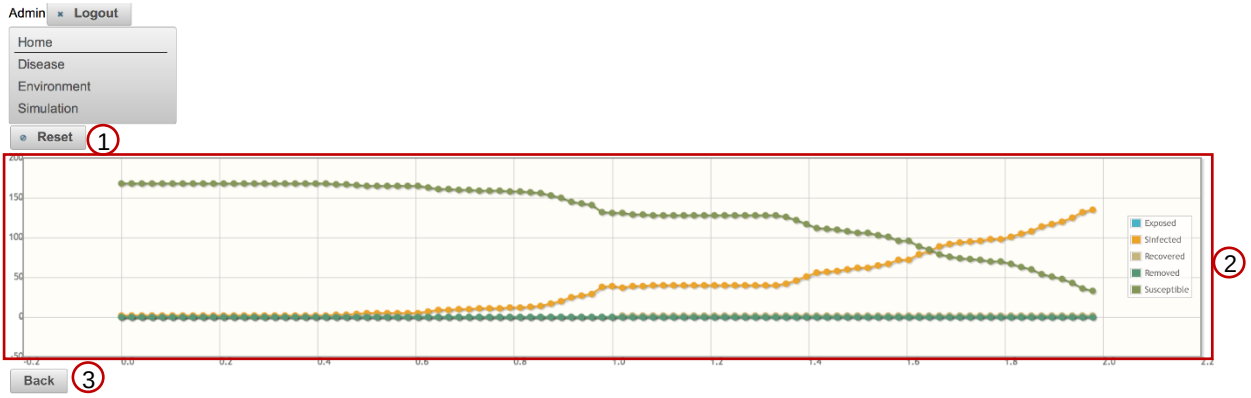

Ilustración 26 - Gráfica estadística de la simulación

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

Como se puede observar, 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.

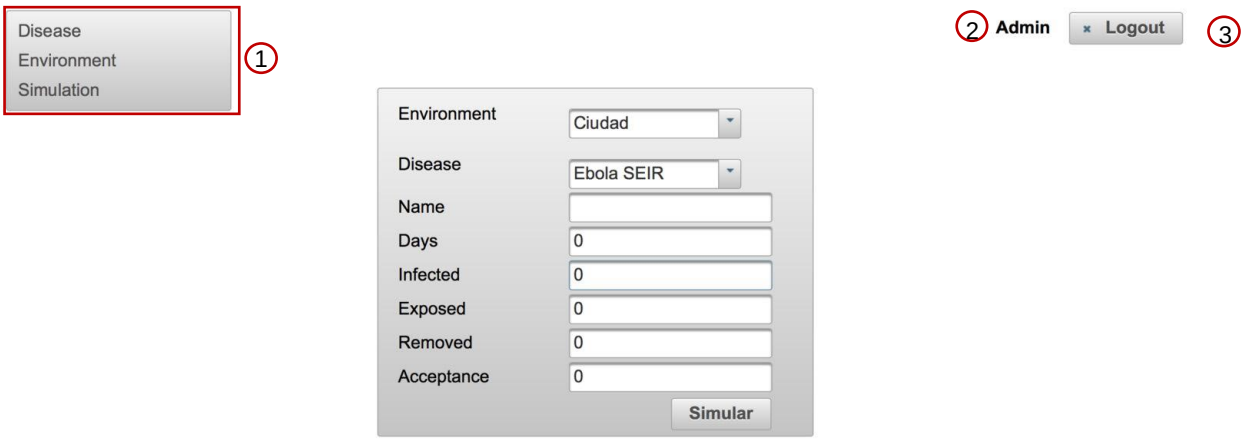

Ilustración 27 - Elementos de la cabecera

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.

b. Manual de usuario de la aplicación de escritorio

Inicialmente se muestra la ventana de Login como se puede ver en la Ilustración 28.

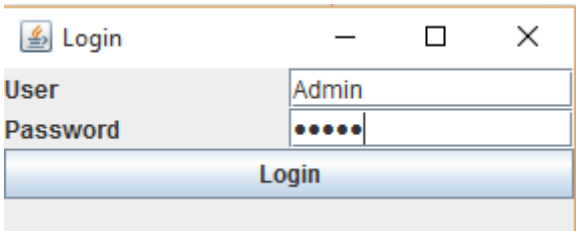

Ilustración 28- Ventana Login

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 29.

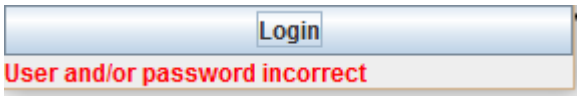

Ilustración 29- Mensaje Login Incorrecto

La ventana principal es en 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 ésta. La ventana principal de la aplicación se puede ver en la Ilustración 30.

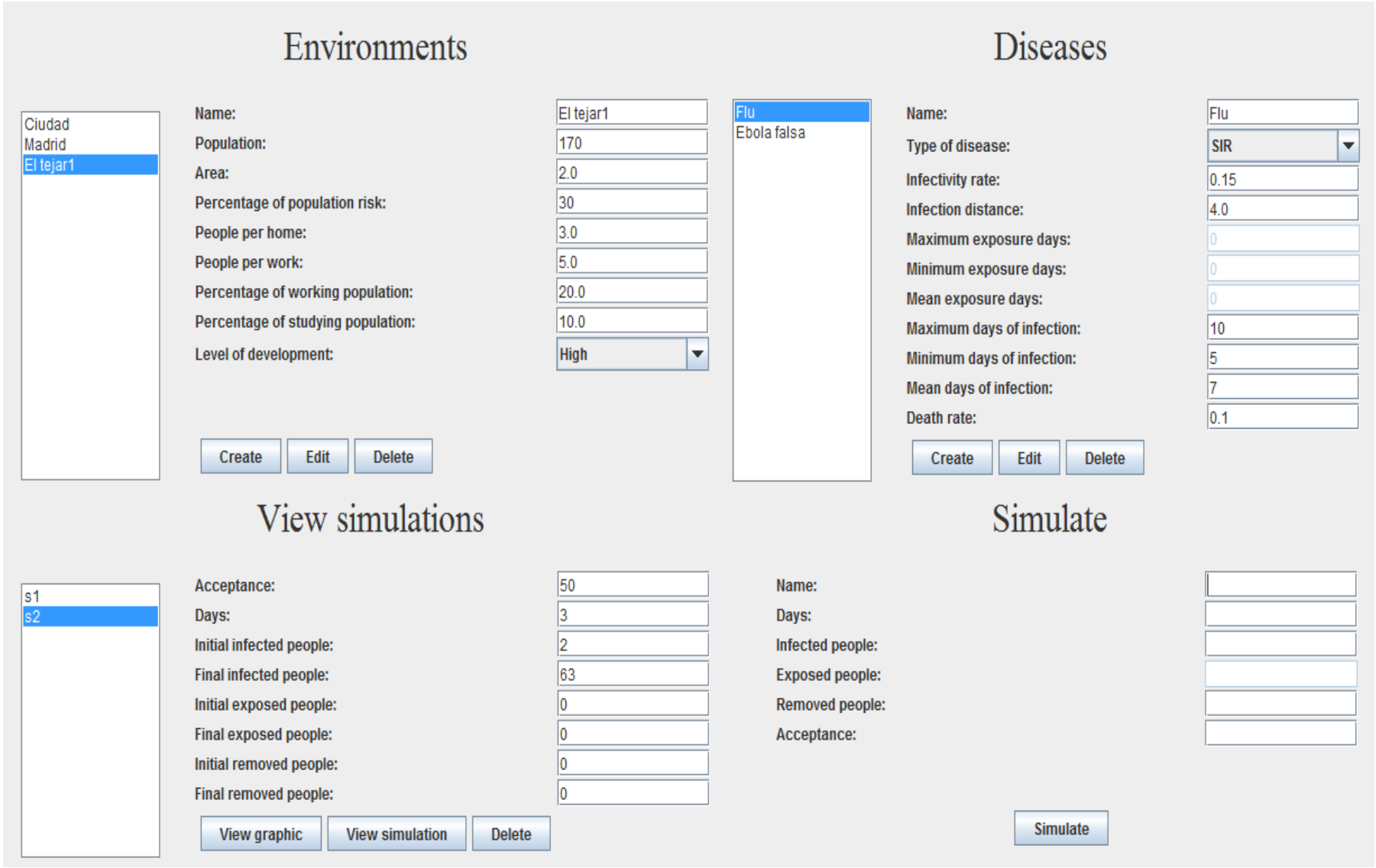

Ilustración 30 - Ventana Principal

La parte de la ventana donde se pueden crear, editar y eliminar un entorno, y a su vez visualizar los datos de éste es la que se puede ver en la Ilustración 31.

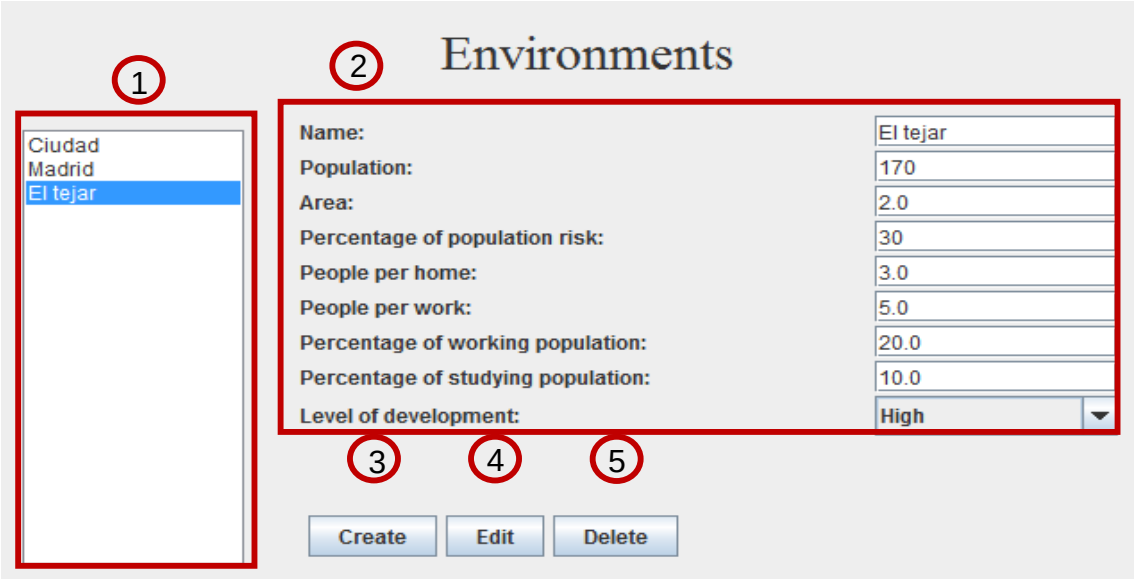

Ilustración 31- Vista de Entornos

1. Lista de entornos: Aquí se muestra un listado con todos los entornos de un usuario. Al hacer doble clic en cualquiera de los de la lista se mostraran 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 1000.
   - Area: Es el área del entorno en km^2. Es un número entre 0 y 30.
   - 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 la población del entorno.
   - People per work: Media de personas por trabajo. Es un número real entre 1 y la población 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 32.

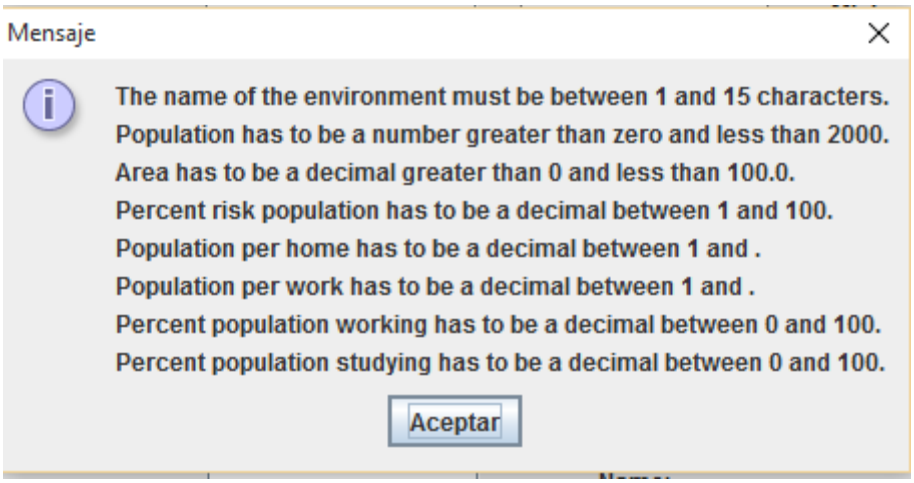

Ilustración 32 - Error de Vista de Entornos

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 33.

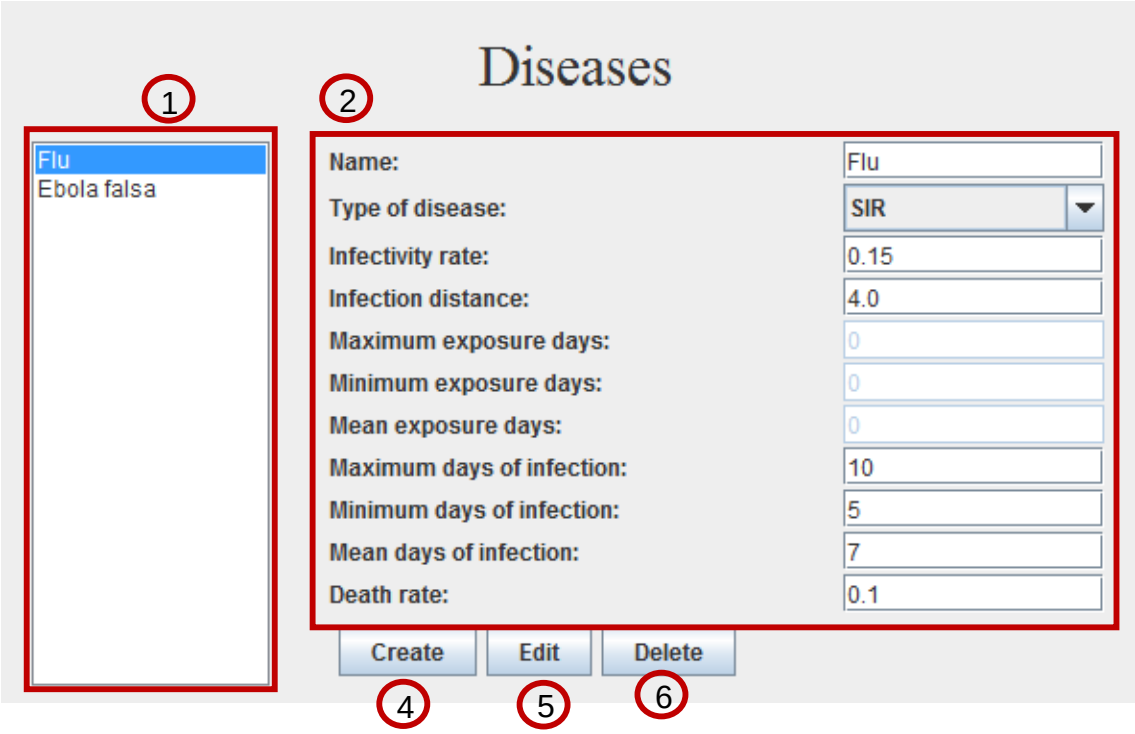

Ilustración 33 - Vista de enfermedades

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 la misma.
2. Datos de la enfermedad: En este formulario se muestran todos los datos de una cierta enfermedad. Este formulario 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 lo son se mostrará su correspondiente mensaje de error, como se puede ver en la Ilustración 34.

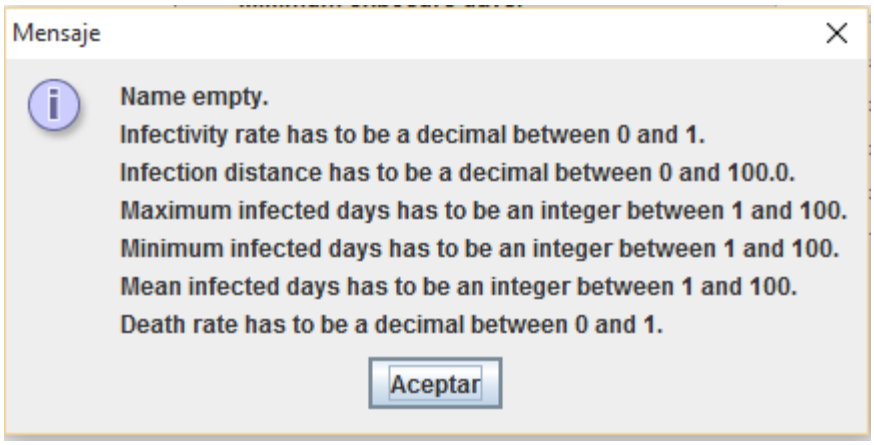

Ilustración 34 - Error de Vista de Enfermedades

La parte de la ventana donde hacer una simulación es la que se puede ver en la Ilustración 35.

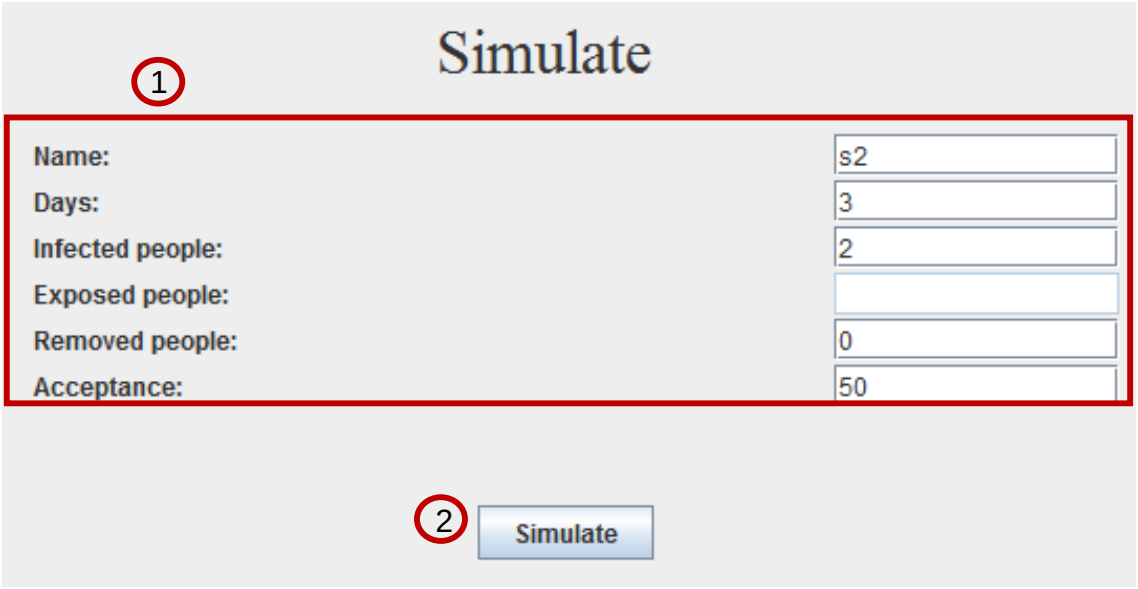

Ilustración 35 - Vista para Hacer Simulación

Primero: Se debe seleccionar una simulación en la lista de simulaciones y un entorno de la lista 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 mínimo de días es 1 y el máximo se establece en función del número de habitantes del entorno seleccionado. Estos máximos son los siguientes:
     - 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 36.

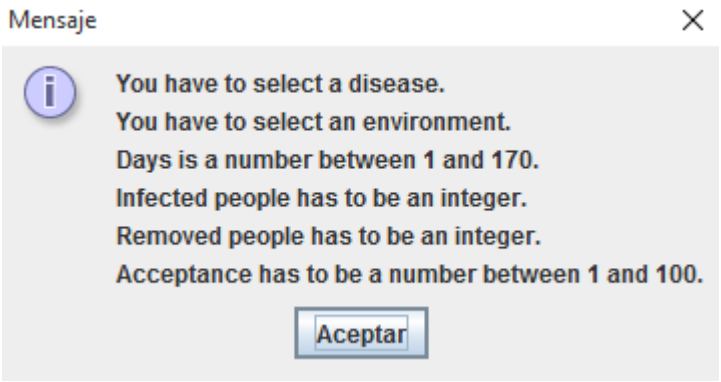

Ilustración 36 - Mensaje de Error Vista Hacer Simulación

La parte de la ventana donde se pueden ver los datos de una simulación es la que se puede ver en la Ilustración 37.

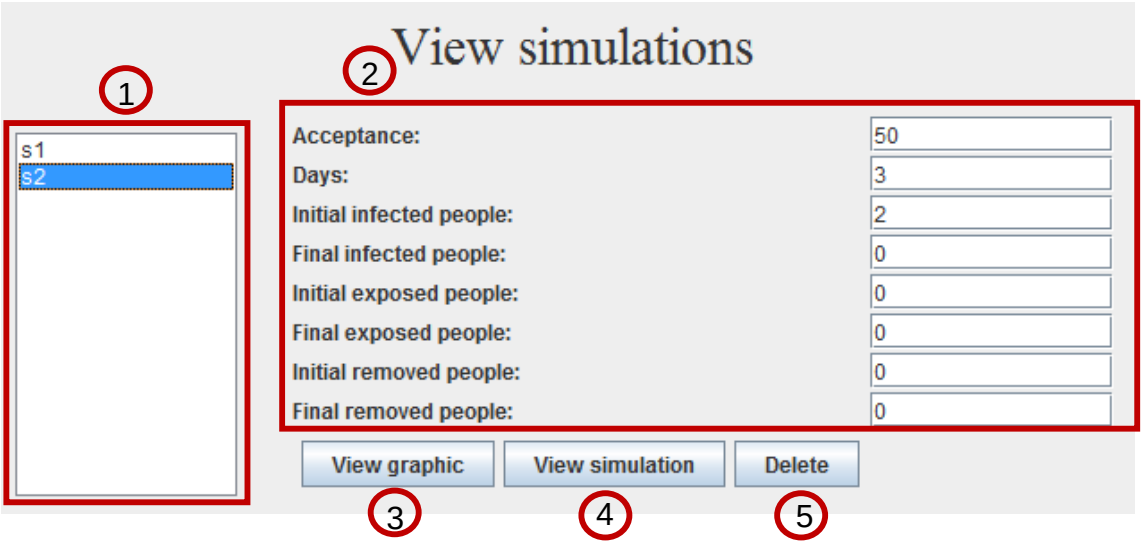

Ilustración 37 - Vista las Simulaciones

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 la misma. 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 visualizar una gráfica de la simulación. Al hacer clic se abre en una ventana aparte como se puede ver en la Ilustración 38.

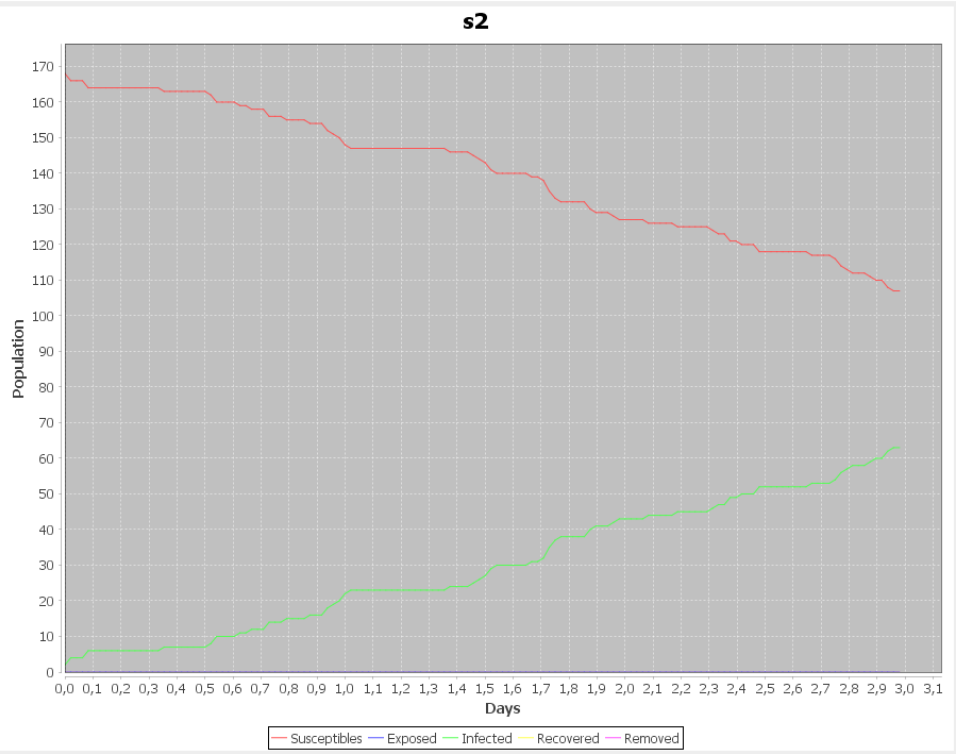

Ilustración 38 - Ventana Grafica de Simulación

4. Ver simulación (View simulation): Muestra una simulación. Al hacer clic se abre en una ventana aparte como se puede ver en la Ilustración 39.

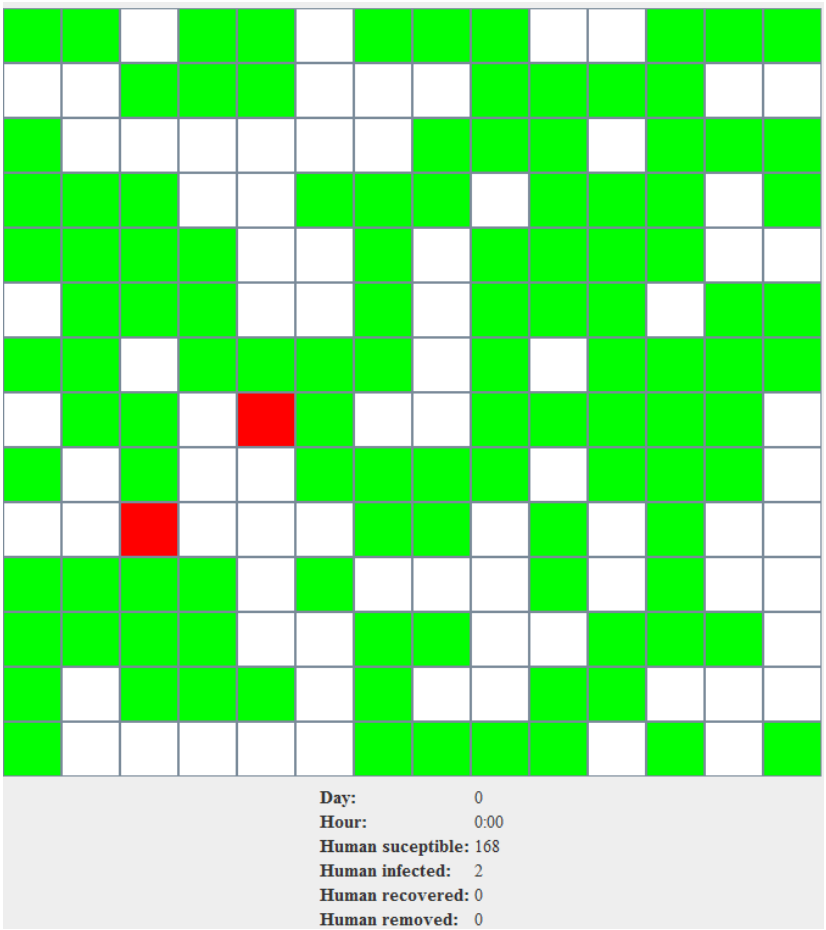

Ilustración 39 - Ventana Ver Simulación

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.

C. Glosario

- **AJAX** (Asynchronous JavaScript And XML). Permite realizar llamadas asíncronas al servidor HTTP evitando recargar la página.
- **API** (Application Programming Interface). Conjunto de métodos que ofrece una librería y permite que se utilicen.
- **AOP** (Agent-Oriented Programming). Paradigma que modela aplicaciones a través de agentes.
- **BDI** (Belief, Desire, Intention). Una de las arquitecturas más populares de los agentes.
- **Licencia BSD** (Licencia Berkley Software Distribution). Licencia software para los sistemas BSD. Es una licencia de software libre permisiva.
- **CRUD** (Create, Read, Update and Delete). Funciones básicas en la base de datos.
- **CSS** (Cascading Style Sheets). Lenguaje encargado de crear la presentación de documentos HTML o XML. los documentos .css son llamados hojas de estilo.
- **FIPA** (Foundation for Intelligent Physical Agents). Fundación que se encarga de elaborar un estándar para los agentes y sistemas multiagente.
- **GLP** (GNU General Public License). Licencia que garantiza la libertad de usar, estudiar, compartir y modificar el software.
- **HTML** (HyperText Markup Language). Lenguaje de marcado para la elaboración de páginas web.
- **HTTP** (Hypertext Transfer Protocol). Protocolo usado en las transacciones de la World Wide Web.
- **JADE** (Java Agent Development Framework). Framework que permite implementar sistemas multiagente.
- **Java EE** (Java Platform Enterprise Edition). Plataforma de programación para desarrollar y ejecutar software de aplicaciones.
- **JSF** (Java Servlet Faces). Tencología de aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario.
- **JSP** (Java Servlet Pages). Tecnología que permite crear páginas web dinámicas basadas en HTML y código Java.
- **JSON** (JavaScript Object Notation). Formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript y su uso principal es en AJAX.
- **MVC** (Model-View-Controller). Patrón de diseño que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.
- **OMS** (Organización Mundial de la Salud). Organismo de las Organización de las Naciones Unidas especializado en gestionar políticas de prevención, promoción e intervención en salud.

- **ORDBMS** (Object-Relational Database Management System). Sistema de gestión de base de datos que sigue un modelo orientado a objetos.
- **RDBMS** (Relational Database Management System). Sistema de gestión de base de datos que sigue el modelo relacional.
- **REST** (Representational State Trasnfer). Interfaz entre sistemas que utiliza HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos.
- **UML** (Unified Modeling Language). Lenguaje de modelado de sistemas de software.
- **XML** (eXtensible Markup Language). Lenguaje de marcas utilizado para almacenar datos en forma legible.