Juego multijugador web de simulación social

Mariano López Muñoz

17 de junio de 2013

Trabajo Fin de Carrera
Ingeniería Técnica en Informática de Gestión

Consultor: Antoni Oller Arcas
Área J2EE

UOC Universitat Oberta de Catalunya

A Sandra y a Diana.
Por soportarme cuando estoy. Por echarme de menos cuando no.

Resumen

Esta memoria final detalla el desarrollo de un motor de juego multijugador web de simulación social.

En primer lugar describiremos el tipo de proyecto que deseamos desarrollar, mostraremos los precedentes que nos llevan a él y acotaremos su alcance. Seguidamente, la temporización que se ha seguido durante el desarrollo del proyecto.

Tras establecer nuestros objetivos y el ritmo a seguir, procederemos al análisis formal de los requisitos, detallando cada uno de los casos de uso que tendremos que satisfacer. También tomaremos el primer contacto con la interfaz de usuario creando un prototipo de las pantallas que la formarán.

Después del análisis elaboraremos el diseño: estableceremos la arquitectura lógica del sistema y mostraremos algunos diagramas de secuencia y el modelo de datos del juego.

Por último, resumiremos los pormenores encontrados durante la fase de implementación, las decisiones tomadas para concluirla satisfactoriamente y las conclusiones finales de todo el trabajo realizado.

Dejaremos en un anexo el código SQL necesario para generar las tablas que utiliza el sistema. Además, en el segundo anexo encontraremos información sobre las primeras pruebas realizadas con el entorno de trabajo, que se realizaron paralelamente a la etapa de planificación, análisis y diseño a modo de toma de contacto para la posterior fase de implementación.

Índice

I Breve descripción del proyecto **1**

1. Precedentes **1**

2. Alcance y motivación **1**

II Planificación **2**

3. Temporización **2**

III Análisis **4**

4. Roles de usuario **4**

5. División en módulos **5**
5.1. Gestión de localizaciones 5
5.2. Gestión de tipos de elemento 6
5.3. Gestión de tipos de trabajo 7
5.4. Gestión del personaje 8
5.5. Motor del sistema 9

6. Descripción de los casos de uso **10**
6.1. Autenticación 10
6.2. Gestión de localizaciones 10
6.3. Gestión de tipos de elemento 12
6.4. Gestión de tipos de trabajo 14
6.5. Gestión del personaje 16
6.6. Motor del sistema 20

7. Entidades **21**

8. Interfaz con el usuario **23**

9. Posibles mejoras **30**

IV Diseño **32**

10. Arquitectura lógica **32**
10.1. Presentación 33
10.2. Negocio 33
Clases controladoras 33
10.3. Persistencia 36
Clases fachada 37

Clases entidad 38

11. Diagramas de secuencia **40**

V Implementación **43**

12. Carencias detectadas **43**

13. Mejoras añadidas **43**

VI Conclusiones **45**

VII Anexo I – Generación de las tablas SQL **46**

VIII Anexo II – Primeras pruebas con el entorno **55**

14. Contenedor **56**

15. Persistencia **57**

16. Anotaciones **58**

17. Resultados **58**

Parte I
Breve descripción del proyecto

Crearemos un sencillo juego de rol multijugador mediante interfaz web. Este permitirá crear un mundo virtual persistente en el que simular la interacción social entre los personajes gestionados por los distintos jugadores. No existirá ninguna meta ni sistemas de puntuación, divisiones en niveles de dificultad, fases, etc... dejando en manos de los jugadores el desarrollo de cada personaje y, tal vez, la creación de una sociedad, economía o historia propios.

1. Precedentes

La idea de este proyecto está fuertemente inspirada en el juego Cantr II[1], que pertenece a la categoría general de los juegos de rol online multijugador masivos (comunmente conocida por sus siglas en inglés: MMORPG[2]), y que sus autores definen más concretamente como "juego de rol persistente mediante navegador web" (PBBRPG[3]).

Aunque se trate de una forma de ocio de escaso interés para el público general, es posible encontrar comunidades de aficionados que muestran gran apego y dedicación de una forma muy estable en el tiempo.

2. Alcance y motivación

Como ya hemos indicado, la interfaz para el usuario será una web. Para ello usaremos componentes HTML relativamente sencillos: textos, botones, tablas, selectores de opciones, etc. Y dejaremos de lado —como posibles mejoras futuras— características más artísticas, como gráficos o componentes más complejos. Nos centraremos en los aspectos de gestión y jugabilidad.

Utilizaremos la tecnología Java EE 6 para realizar un juego simple pero altamente ampliable, mantenible y personalizable; de forma que sea más importante el motor de juego y las posibilidades que nos permita, que la demostración concreta con la que podamos probar el software.

La intención final será publicar este trabajo bajo alguna licencia de código abierto que permita —en un futuro más allá del alcance de este proyecto— la colaboración directa de la comunidad para, idealmente, sumar esfuerzos en su posterior desarrollo. Intentaremos superar las dificultades y reticencias que normalmente supone, en el caso concreto de los juegos, su publicación abierta. Por ejemplo, el miedo a las grietas de seguridad que permitan a jugadores malintencionados saltarse las reglas o acceder a datos sensibles, o una posible fragmentación entre varios hipotéticos servidores que implementen el mismo motor y compitan entre sí.

[1] http://www.cantr.net

[2] Acrónimo de *massively multiplayer online role-playing game*.

[3] Acrónimo de *persistent browser-based role-playing game*.

Parte II
Planificación

Dividiremos las tareas según cuatro hitos o entregas:

Plan de trabajo: primer documento con la descripción inicial del proyecto, división en tareas y temporización del mismo.

PEC 2: entrega parcial que incluirá una versión previa de la memoria con el análisis de requerimientos y el diseño funcional.

PEC 3: entrega parcial que incluirá una versión funcional del software y avanza en la elaboración de la memoria.

Memoria y presentación: entrega final con la memoria del proyecto, la presentación virtual y el producto finalizado.

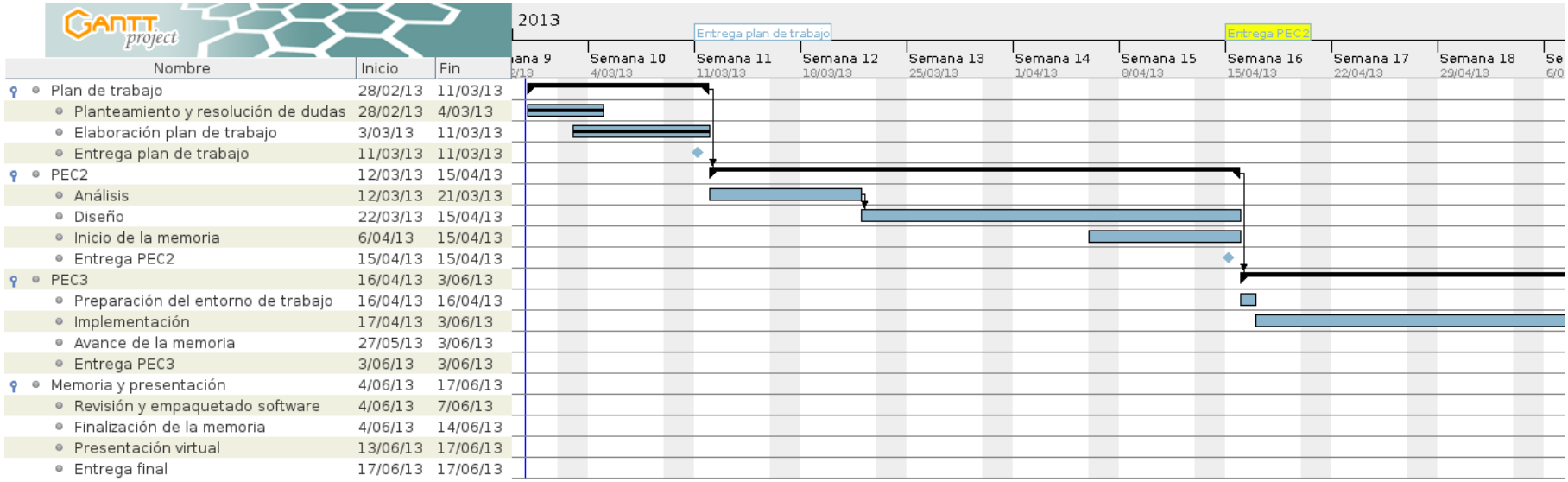
Hito / tarea	Duración (días)	Fecha de entrega
Plan de trabajo	12	11/03/2013
Planteamiento y resolución de dudas	4	
Elaboración del plan	8	
PEC 2	35	15/04/2013
Análisis	10	
Diseño	20	
Inicio de la memoria	5	
PEC 3	49	03/06/2013
Preparación del entorno de trabajo	1	
Implementación	44	
Avance de la memoria	4	
Memoria y presentación	14	17/06/2013
Revisión y empaquetado del software	2	
Finalización de la memoria	8	
Presentación virtual	4	

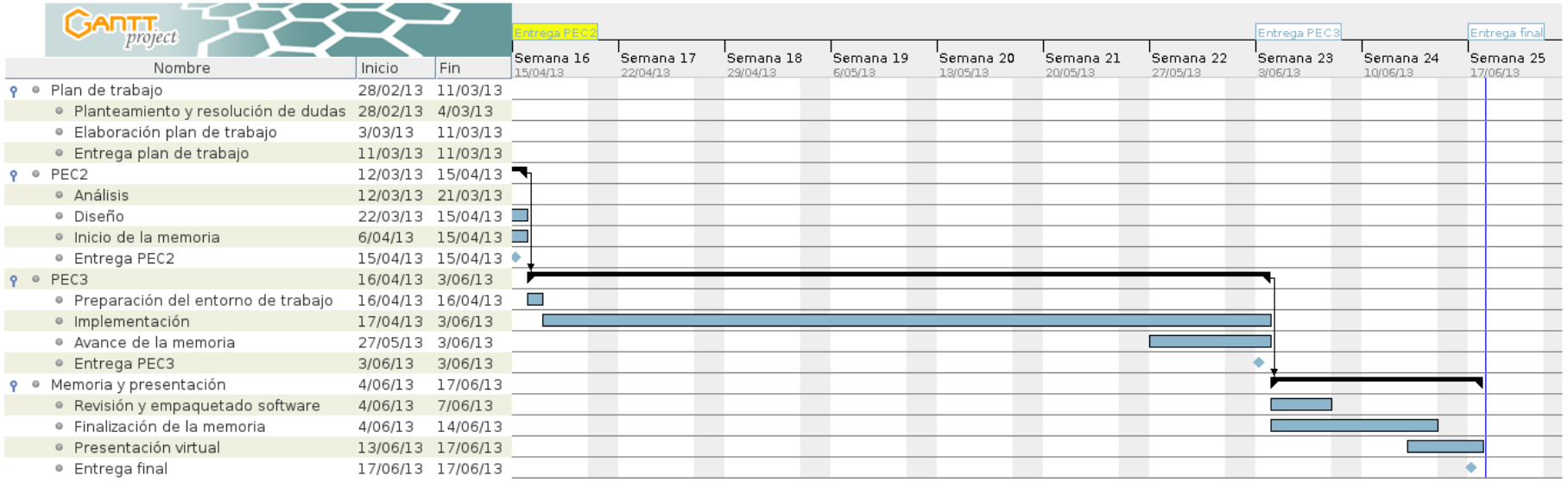
3. Temporización

Una vez conocidas las fechas de entrega y la estimación de la duración total de cada tarea, mostraremos un diagrama de Gantt con la temporización de las mismas.

Para intentar hacer una planificación un poco más realista, hemos tenido en cuenta posibles solapamientos en el tiempo entre determinadas tareas. Por ejemplo, la elaboración de algunos documentos a la par que terminamos otras tareas técnicas. Esto implica que, cuando dos tareas se solapan en el tiempo, estamos invirtiendo una dedicación del 50 % a cada una de ellas.

Por motivos de espacio, y para mayor claridad, en la página siguiente se muestra el diagrama de Gantt en orientación apaisada y dividido en dos partes: antes y después de la entrega de la PEC 2.


GANTT project
2013
Entrega plan de trabajo
Entrega PEC2
Semana 9
2/3
Semana 10
4/03/13
Semana 11
11/03/13
Semana 12
18/03/13
Semana 13
25/03/13
Semana 14
1/04/13
Semana 15
8/04/13
Semana 16
15/04/13
Semana 17
22/04/13
Semana 18
29/04/13
Se
6/0
Nombre
Inicio
Fin
Plan de trabajo 28/02/13 11/03/13
Planteamiento y resolución de dudas 28/02/13 4/03/13
Elaboración plan de trabajo 3/03/13 11/03/13
Entrega plan de trabajo 11/03/13 11/03/13
PEC2 12/03/13 15/04/13
Análisis 12/03/13 21/03/13
Diseño 22/03/13 15/04/13
Inicio de la memoria 6/04/13 15/04/13
Entrega PEC2 15/04/13 15/04/13
PEC3 16/04/13 3/06/13
Preparación del entorno de trabajo 16/04/13 16/04/13
Implementación 17/04/13 3/06/13
Avance de la memoria 27/05/13 3/06/13
Entrega PEC3 3/06/13 3/06/13
Memoria y presentación 4/06/13 17/06/13
Revisión y empaquetado software 4/06/13 7/06/13
Finalización de la memoria 4/06/13 14/06/13
Presentación virtual 13/06/13 17/06/13
Entrega final 17/06/13 17/06/13


GANTT project
Entrega PEC2
Entrega PEC3
Entrega final
Semana 16
15/04/13
Semana 17
22/04/13
Semana 18
29/04/13
Semana 19
6/05/13
Semana 20
13/05/13
Semana 21
20/05/13
Semana 22
27/05/13
Semana 23
3/06/13
Semana 24
10/06/13
Semana 25
17/06/13
Nombre
Inicio
Fin
Plan de trabajo 28/02/13 11/03/13
Planteamiento y resolución de dudas 28/02/13 4/03/13
Elaboración plan de trabajo 3/03/13 11/03/13
Entrega plan de trabajo 11/03/13 11/03/13
PEC2 12/03/13 15/04/13
Análisis 12/03/13 21/03/13
Diseño 22/03/13 15/04/13
Inicio de la memoria 6/04/13 15/04/13
Entrega PEC2 15/04/13 15/04/13
PEC3 16/04/13 3/06/13
Preparación del entorno de trabajo 16/04/13 16/04/13
Implementación 17/04/13 3/06/13
Avance de la memoria 27/05/13 3/06/13
Entrega PEC3 3/06/13 3/06/13
Memoria y presentación 4/06/13 17/06/13
Revisión y empaquetado software 4/06/13 7/06/13
Finalización de la memoria 4/06/13 14/06/13
Presentación virtual 13/06/13 17/06/13
Entrega final 17/06/13 17/06/13

Parte III
Análisis

Vamos a analizar los requisitos funcionales de nuestro proyecto. Comenzaremos por definir a grandes rasgos los diferentes tipos de usuario que van a interactuar con el sistema. Después razonaremos los módulos en los que vamos a dividirlo y los elementos que los componen, y describiremos los distintos casos de uso detectados. Por último, abordaremos la interfaz con el usuario.

4. Roles de usuario

En general, existirán dos perfiles de usuario en el sistema: administradores y jugadores.

Administrador: Los administradores se encargarán de gestionar el "mundo virtual" dentro del cual se desarrolla todo el juego. Será necesario crear y gestionar las localizaciones que pueden visitar los personajes, indicar los recursos de los que estas disponen y definir los tipos de alimentos, materiales o herramientas de los que se podrá hacer uso, así como los trabajos que pueden realizar los personajes y los productos que obtendrán.

Jugador: Un jugador podrá crear varios personajes y elegir cuál de ellos gestiona en un momento dado. Del personaje podrá consultar su estado: características, habilidades, salud...; así como la información de la localización donde se encuentre: listado de otros personajes que están presentes, los objetos disponibles, descripción de los eventos que hayan sucedido o los caminos de salida. También podrá realizar una serie de acciones, como decir alguna frase, recoger un objeto, comenzar un trabajo o desplazarse a otra localización.

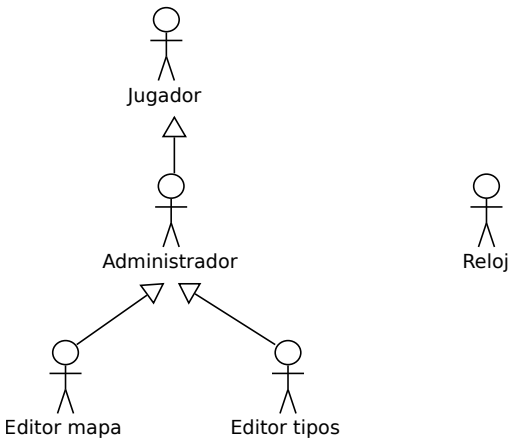


Figura 1: Identificación y relación entre los actores del sistema.

En el caso concreto de los administradores, nos puede interesar que existan algunos usuarios capaces de mantener los tipos de elemento y de trabajo, pero que no tengan

acceso al mapa del juego. Y viceversa: algunos administradores tendrían permiso para mantener las localizaciones y sus recursos disponibles, pero sin poder definir elementos ni trabajos.

Por otra parte, todo usuario del sistema necesitará un nombre de usuario y contraseña para autenticarse y acceder al mismo. Por simplicidad, obviaremos este requerimiento en los diagramas de casos de uso.

5. División en módulos

Al describir los distintos tipos de usuario del sistema y sus funciones, podemos hacernos una primera idea de cómo dividir el sistema en módulos. En total hemos identificado cinco módulos con responsabilidades bien diferenciadas.

Los tres primeros permitirán a los administradores definir con el máximo detalle el mundo virtual donde se van a mover los personajes. Los dos últimos permitirán llevar a cabo el proceso de juego en sí mismo, por parte de los jugadores y del sistema.

5.1. Gestión de localizaciones

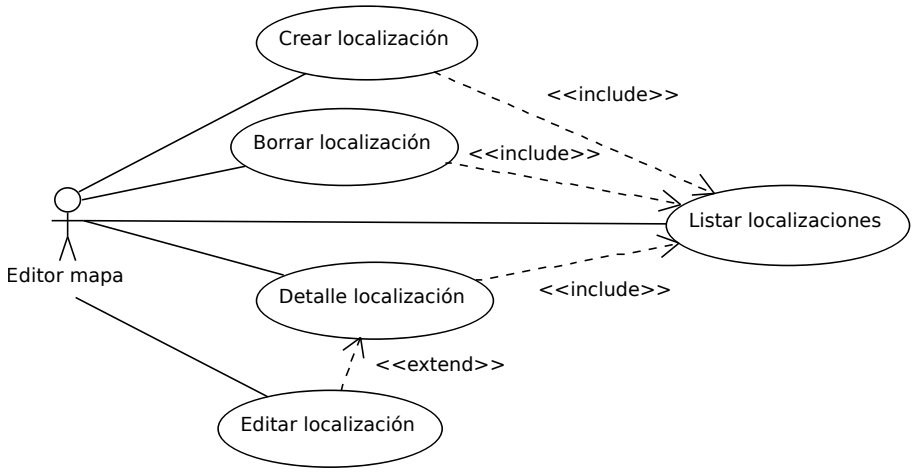


Figura 2: Diagrama de casos de uso. Gestión de localizaciones.

Para el desarrollo del juego deberemos contar con un mundo virtual por donde los personajes puedan ir moviéndose y realizando distintas acciones. Este mundo lo dividiremos en localizaciones, de forma que cada personaje puede estar en una única localización en un momento dado.

Cada localización estará conectada a otras localizaciones mediante un camino. Idealmente todas las localizaciones deberían estar conectadas entre sí, directa o indirectamente, formando una estructura de grafo conexo.

Para cada localización también podremos elegir, de entre los existientes en el sistema, los recursos naturales que tendrá disponibles. Además se le podrá asignar un nombre y le corresponderán unas coordenadas concretas que la sitúen dentro del mapa del juego.

El módulo de gestión de localizaciones permitirá a un administrador listar las localizaciones existentes, así como borrarlas, ver su detalle, editarlas o crear una nueva.

5.2. Gestión de tipos de elemento

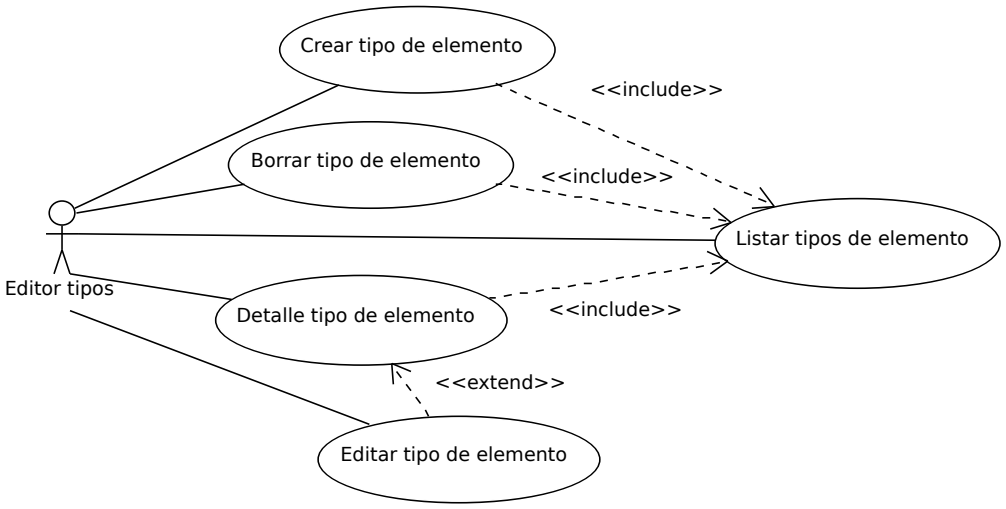


Figura 3: Diagrama de casos de uso. Gestión de tipos de elemento.

Un elemento es cualquier volumen, cuerpo u objeto que los personajes pueden coger, llevar, manipular o usar con distintas finalidades. Se deberá indicar al sistema los tipos de elemento que se podrán crear durante el juego, ya que todo elemento existente dentro del mundo virtual debe corresponderse con exactamente uno de los tipos que definamos.

A su vez, cada tipo de elemento pertenecerá a una o varias categorías generales. Cada categoría otorga al tipo de elemento unas determinadas características y comportamientos.

Recurso: un recurso es un tipo de elemento proporcionado por la naturaleza u otras circunstancias especiales de una localización concreta. Cada localización puede tener asignados varios recursos que los personajes pueden recolectar o extraer. Para cada uno de ellos se pueden definir las herramientas necesarias para obtenerlo, así como la tasa media de extracción con cada una de ellas.

Herramienta: Cuando la presencia de un tipo de elemento es necesaria para realizar algún tipo de trabajo, el primero se considerará una herramienta.

Alimento: Un alimento es un elemento que los personajes pueden ingerir, produciendo algún efecto en este y que desaparece en el proceso. Especificaremos el efecto de saciedad del hambre, restauración de salud o envenenamiento que produce sobre el personaje.

Material: Llamaremos material al elemento necesario para la realización de algún tipo de trabajo, y que es modificado o desaparece como tal tras la consecución del mismo.

Arma: Cuando un personaje ataca a otro, ambos podrán aprovechar las capacidades de las armas que lleven en ese momento. Consideraremos que es un arma cualquier elemento que ayude a hacer más daño durante el ataque o, por el contrario, sirva para reducir o anular los efectos del mismo cuando el sujeto se defiende. Cada arma tendrá asignado un valor de ataque y otro de defensa.

El módulo de gestión de tipos de elemento permitirá al administrador listar los tipos ya existentes, borrar, ver el detalle o crear uno nuevo.

Además, como excepción, el sistema tendrá predefinido un tipo de elemento especial: las notas. Este tipo permanecerá fijo en el sistema, de forma que no podrá borrarse ni editarse por los administradores. Por este motivo, tampoco será necesario que aparezca en el listado:

Nota: Una nota permite a los personajes escribir un texto en ella, o bien leerlo. Consideraremos que no es necesario ningún material o herramienta para crear una nota. Un personaje podrá leer una nota que tenga en su inventario.

5.3. Gestión de tipos de trabajo

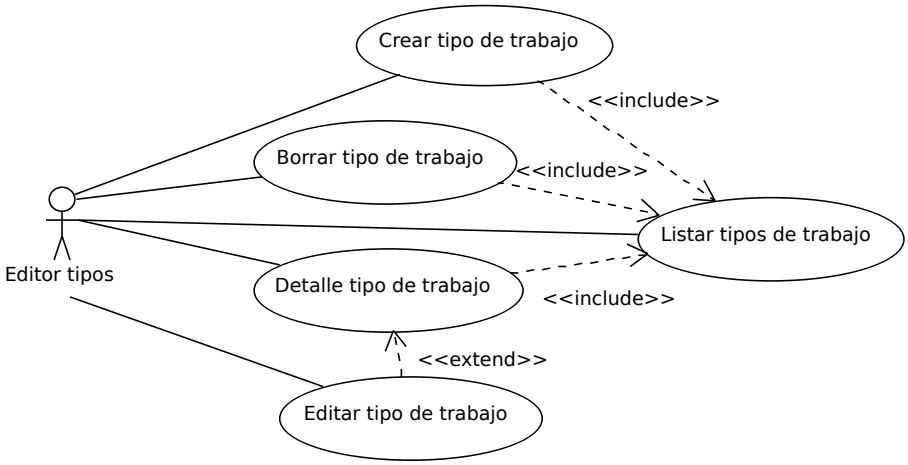


Figura 4: Diagrama de casos de uso. Gestión de tipos de trabajo.

Un trabajo será una actividad que desarrolla un personaje y que tiene una duración determinada. Un trabajo puede requerir algunos materiales de entrada y herramientas para poder realizarlo; y dará como resultado un nuevo elemento. De esta forma, para cada tipo de trabajo tendremos que seleccionar distintos tipos de elemento que lo componen:

- Una lista de materiales. Para cada uno de los materiales indicaremos la cantidad necesaria, expresada en unidades o gramos según corresponda.
- Una lista de herramientas, que deben tener en su inventario el o los personajes que estén realizando el trabajo en ese momento.
- Un tipo de elemento de salida. Indicaremos la cantidad que se produce para las cantidades de entrada indicadas.

La lista de herramientas puede estar vacía. Pero al menos debe existir un material, y siempre existirá exactamente una salida.

Un trabajo también deben especificar las habilidades que se requieren para ejecutarlo satisfactoriamente y el número de turnos que, en condiciones normales (participando un solo personaje que tiene las habilidades especificadas), se tardará en concluirlo. Un personaje que iguale o supere todas las habilidades necesarias podrá realizar el trabajo en el tiempo marcado e incluso un poco antes; mientras que un personaje que no llegue a estos mínimos tardará más. Este tiempo se alargará más cuanto más alejados se encuentren —del mínimo necesario— una o varias de las habilidades del personaje. Por otra parte, el trabajo concluirá antes cuantos más personajes participen en él.

Como caso especial de tipo de trabajo, tenemos la extracción de recursos. Este tipo de trabajo será fijo en el sistema y no será necesario su mantenimiento.

De forma análoga a los módulos anteriores, la gestión de tipos de trabajo permite el mismo grupo de funcionalidades *crud* sobre los tipos de trabajo; y solamente puede ser realizada por los administradores.

5.4. Gestión del personaje

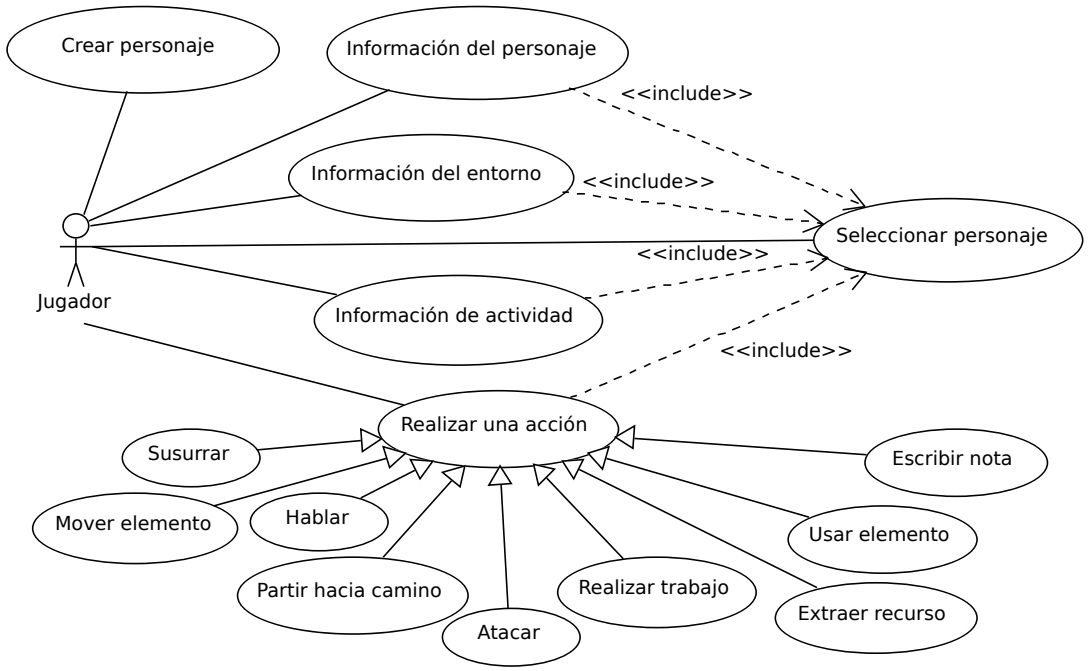


Figura 5: Diagrama de casos de uso. Gestión del personaje.

El módulo de gestión del personaje permitirá a los jugadores ejercer su labor como tal. Es decir, experimentar el juego.

Un jugador puede crear los personajes que desee hasta un máximo de diez. Cuando el jugador decide crear un personaje, el sistema se encarga de asignarle unas determinadas habilidades y una localización de inicio, de forma aleatoria. Una vez que el jugador disponga de uno o más personajes, podrá gestionar —jugar interpretando su rol— cada

uno de ellos de forma independiente. Para esto, primero debe seleccionar el personaje con el que desea jugar en ese momento.

Una vez el jugador tiene un personaje seleccionado, podrá realizar múltiples acciones distintas sobre él. Si hablamos desde la perspectiva del personaje, este puede consultar su propio estado: salud, habilidades, inventario de objetos que lleva en ese instante, nombre por el que se conoce a sí mismo y su edad. También podrá consultar la información acerca del entorno en el que se encuentra: recursos disponibles, personajes y elementos que tiene al alcance en la localización donde se encuentra, posibles caminos para viajar, así como una descripción de los eventos ocurridos.

Un personaje también dispondrá de un amplio abanico de acciones a realizar. Algunas de ellas tendrán un efecto casi inmediato en el entorno: hablar, atacar a otro personaje, o mover un elemento de lugar (cogerlo, soltarlo, dárselo a otro personaje). Otras acciones requerirán el paso de uno o varios turnos de juego para que el cambio ocurra: viajar hacia otra localización, avanzar un trabajo o extraer algún recurso.

5.5. Motor del sistema

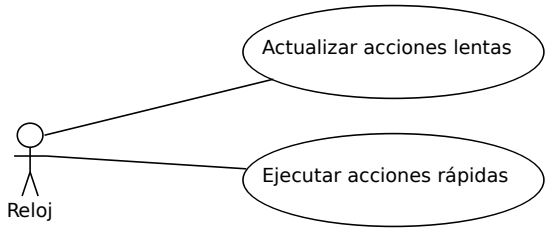


Figura 6: Diagrama de casos de uso. Motor del sistema.

Este módulo se encarga de controlar el transcurso del tiempo en forma de turnos, y gestionar el resultado de los eventos que hayan sucedido.

Cada determinado intervalo de tiempo, se producirá el paso de un turno de juego al siguiente. Cada vez que esto sucede, se recorrerán todos las acciones lentas pendientes y se actualizará el estado de cada una de ellas, comprobando qué trabajos han terminado. Se mostrará, si fuera necesario, un aviso a los personajes implicados o que presencien cada uno de los eventos que se produzca. Cada vez que pase un turno, se restaurará parte del cansancio que cada personaje tuviera acumulado.

También de forma periódica, aunque asíncrona y lo más inmediata posible, se recorrerán las acciones rápidas que los jugadores han ejecutado, con el fin de actualizar y mostrar los eventos pertinentes.

Este módulo eliminará de base de datos los elementos consumidos durante un trabajo, creará los nuevos productos obtenidos, actualizará estado de estos y de los personajes, etc.

6. Descripción de los casos de uso

Realizaremos una descripción detallada de cada uno de los casos de uso detectados. Para cada uno de ellos mostraremos una tabla con el nombre del caso de uso en cuestión, seguido de la información pertinente. Ordenaremos las tablas según el módulo o funcionalidad implicados.

6.1. Autenticación

Nombre	**Autenticar**
Actor	Jugador, administrador
Precondición	-
Escenario	1. Se muestra la página "Autenticar". 2. El usuario introduce los datos: nombre de usuario y contraseña. 3. El usuario pulsa "Entrar". 4. Se comprueba que los datos son correctos. 5. Se muestra la página "Inicio".
Alternativo	4a. Existe algún dato erróneo y se muestra un mensaje de error. 4a.1. Vuelve a 2.
Resultado	El usuario se encuentra autenticado

6.2. Gestión de localizaciones

Nombre	**Listar localizaciones**
Actor	Editor mapa
Precondición	El usuario se encuentra autenticado
Escenario	1. Se muestra el menú del administrador. 2. El usuario pulsa "Gestionar mapa". 3. Se muestra la pantalla "Gestión del mapa".
Alternativo	-
Resultado	-

Nombre	**Crear localización**
Actor	Editor mapa
Precondición	Se ha realizado el c.u. "Listar localizaciones"
Escenario	1. El usuario pulsa "Crear localización". 2. Se muestra la pantalla "Detalle de localización" vacía. 3. El usuario introduce los datos: nombre de la localización, coordenadas, lugares conectados y recursos disponibles. 4. El usuario pulsa "Guardar". 5. Se guarda la nueva localización y se muestra un mensaje de éxito. 6. Se muestra la pantalla "Gestión del mapa".
Alternativo	3a. En cualquier momento el usuario pulsa "Cancelar". 3a.1. Se muestra la pantalla "Gestión del mapa". 5a. Existe algún dato erróneo y se muestra un mensaje de error. 5a.1. Vuelve a 3.
Resultado	La nueva localización queda registrada

Nombre	**Borrar localización**
Actor	Editor mapa
Precondición	Se ha realizado el c.u. "Listar localizaciones" y se lista al menos una
Escenario	1. El usuario pulsa "Borrar" sobre una de las listadas. 2. Se comprueba que la localización esté vacía. 3. Se muestra un mensaje de aviso con las opciones "Borrar" y "Cancelar". 4. El usuario pulsa "Borrar". 5. Se borra la localización y se muestra un mensaje de éxito.
Alternativo	4a. El usuario pulsa "Cancelar". 2a. Se comprueba que la localización no está vacía. 2a.1. Se muestra un mensaje de error.
Resultado	La localización se borra del registro

Nombre	**Detalle localización**
Actor	Editor mapa
Precondición	Se ha realizado el c.u. "Listar localizaciones" y se lista al menos una
Escenario	1. El usuario pulsa "Detalle" sobre una de las listadas. 2. Se muestra la pantalla "Detalle de localización" con los datos de la misma. 3. El usuario pulsa "Cancelar". 4. Se muestra la pantalla "Gestión del mapa".
Alternativo	-
Resultado	-

Nombre	**Editar localización**
Actor	Editor mapa
Precondición	Se ha realizado el c.u. "Listar localizaciones" y se lista al menos una
Escenario	1. El usuario pulsa "Detalle localización" sobre una de las listadas. 2. Se muestra la pantalla "Detalle de localización" con los datos de la misma. 3. El usuario modifica algún dato. 4. El usuario pulsa "Guardar". 5. Se guardan los cambios y se muestra un mensaje de éxito. 6. Se muestra la pantalla "Gestión del mapa".
Alternativo	5a. Existe algún dato erróneo y se muestra un mensaje de error. 5a.1. Vuelve a 3.
Resultado	Se actualiza la localización en el registro

6.3. Gestión de tipos de elemento

Nombre	**Listar tipos de elemento**
Actor	Editor tipos
Precondición	El usuario se encuentra autenticado
Escenario	1. Se muestra el menú del administrador. 2. El usuario pulsa "Gestionar tipos de elemento". 3. Se muestra la pantalla "Gestión de tipos de elemento".
Alternativo	-
Resultado	-

Nombre	**Crear tipo de elemento**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de elemento"
Escenario	1. El usuario pulsa "Crear tipo de elemento". 2. Se muestra la pantalla "Detalle de tipos de elemento" vacía. 3. El usuario introduce los datos: nombre y categorías a la que pertenece. 4. El usuario pulsa "Guardar". 5. Se guarda el nuevo tipo y se muestra un mensaje de éxito. 6. Se muestra la pantalla "Gestión de tipos de elemento".
Alternativo	3a. En cualquier momento el usuario pulsa "Cancelar". 3a.1. Se muestra la pantalla "Gestión de tipos de elemento". 4a. Una de las categorías es "Recurso" y el usuario introduce los datos específicos: listado de herramientas y tasa de extracción. 4a.1. Vuelve a 4. 4b. Una de las categorías es "Alimento" y el usuario introduce los datos específicos: saciedad y curación o envenenamiento. 4b.1. Vuelve a 4. 4c. Una de las categorías es "Arma" y el usuario introduce los datos específicos: ataque y defensa. 4c.1. Vuelve a 4. 5a. Existe algún dato erróneo y se muestra un mensaje de error. 5a.1. Vuelve a 3.
Resultado	El nuevo tipo de elemento queda registrado

Nombre	**Borrar tipo de elemento**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de elemento" y se lista al menos uno
Escenario	1. El usuario pulsa "Borrar" sobre uno de los tipos listados. 2. Se comprueba que no existan instancias de este tipo. 3. Se comprueba que no existan tipos de trabajo que usen este tipo. 4. Se muestra un mensaje de aviso con las opciones "Borrar" y "Cancelar". 5. El usuario pulsa "Borrar". 6. Se borra el tipo de elemento y se muestra un mensaje de éxito.
Alternativo	4a. El usuario pulsa "Cancelar". 2a. Se comprueba que existen instancias o que algún tipo de trabajo usa este tipo. 2a.1. Se muestra un mensaje de error.
Resultado	El tipo de elemento se borra del registro

Nombre	**Detalle tipo de elemento**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de elemento" y se lista al menos uno
Escenario	1. El usuario pulsa "Detalle" sobre uno de los tipos listados. 2. Se muestra la pantalla "Detalle de tipo de elemento" con los datos del mismo. 3. El usuario pulsa "Cancelar". 4. Se muestra la pantalla "Gestión de tipos de elemento".
Alternativo	-
Resultado	-

Nombre	**Editar tipo de elemento**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de elemento" y se lista al menos uno
Escenario	1. El usuario pulsa "Detalle" sobre uno de los tipos listados. 2. Se muestra la pantalla "Detalle de tipo de elemento" con los datos del mismo. 3. El usuario modifica algún dato. 4. El usuario pulsa "Guardar". 5. Se guardan los cambios y se muestra un mensaje de éxito. 6. Se muestra la pantalla "Gestión de tipos de elemento".
Alternativo	5a. Existe algún dato erróneo y se muestra un mensaje de error. 5a.1. Vuelve a 3.
Resultado	Se actualiza el tipo de elemento en el registro

6.4. Gestión de tipos de trabajo

Nombre	**Listar tipos de trabajo**
Actor	Editor tipos
Precondición	El usuario se encuentra autenticado
Escenario	1. Se muestra el menú del administrador. 2. El usuario pulsa "Gestionar tipos de trabajo". 3. Se muestra la pantalla "Gestión de tipos de trabajo".
Alternativo	-
Resultado	-

Nombre	**Crear tipo de trabajo**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de trabajo"
Escenario	1. El usuario pulsa "Crear tipo de trabajo". 2. Se muestra la pantalla "Detalle de tipos de trabajo" vacía. 3. El usuario introduce los datos: nombre, duración, habilidades necesarias, materiales, herramientas y salida. 4. El usuario pulsa "Guardar". 5. Se guarda el nuevo tipo y se muestra un mensaje de éxito. 6. Se muestra la pantalla "Gestión de tipos de trabajo".
Alternativo	3a. En cualquier momento el usuario pulsa "Cancelar". 3a.1. Se muestra la pantalla "Gestión de tipos de trabajo". 5a. Existe algún dato erróneo y se muestra un mensaje de error. 5a.1. Vuelve a 3.
Resultado	El nuevo tipo de trabajo queda registrado

Nombre	**Borrar tipo de trabajo**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de trabajo" y se lista al menos uno
Escenario	1. El usuario pulsa "Borrar" sobre uno de los tipos listados. 2. Se comprueba que no existan instancias de este tipo en proceso. 3. Se muestra un mensaje de aviso con las opciones "Borrar" y "Cancelar". 4. El usuario pulsa "Borrar". 5. Se borra el tipo de trabajo y se muestra un mensaje de éxito.
Alternativo	4a. El usuario pulsa "Cancelar". 2a. Se comprueba que existen instancias en proceso. 2a.1. Se muestra un mensaje de error.
Resultado	El tipo de trabajo se borra del registro

Nombre	**Detalle tipo de trabajo**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de trabajo" y se lista al menos uno
Escenario	1. El usuario pulsa "Detalle" sobre uno de los tipos listados. 2. Se muestra la pantalla "Detalle de tipo de trabajo" con los datos del mismo. 3. El usuario pulsa "Cancelar". 4. Se muestra la pantalla "Gestión de tipos de trabajo".
Alternativo	-
Resultado	-

Nombre	**Editar tipo de trabajo**
Actor	Editor tipos
Precondición	Se ha realizado el c.u. "Listar tipos de trabajo" y se lista al menos uno
Escenario	1. El usuario pulsa "Detalle" sobre uno de los tipos listados. 2. Se muestra la pantalla "Detalle de tipo de trabajo" con los datos del mismo. 3. El usuario modifica algún dato. 4. El usuario pulsa "Guardar". 5. Se guardan los cambios y se muestra un mensaje de éxito. 6. Se muestra la pantalla "Gestión de tipos de trabajo".
Alternativo	5a. Existe algún dato erróneo y se muestra un mensaje de error. 5a.1. Vuelve a 3.
Resultado	Se actualiza el tipo de trabajo en el registro

6.5. Gestión del personaje

Nombre	**Crear personaje**
Actor	Jugador
Precondición	El usuario se encuentra autenticado
Escenario	1. Se muestra el menú del jugador. 2. El usuario pulsa "Crear personaje". 3. Se comprueba que el usuario tenga menos de 10 personajes creados. 4. Se crea un nuevo personaje con características aleatorias. 5. Se guarda el personaje y se muestra un mensaje de éxito.
Alternativo	3a. Se comprueba que el usuario ya tiene 10 personajes creados. 3a.1. Se muestra un mensaje de aviso. 3a.2. Vuelve a 1.
Resultado	El nuevo personaje queda registrado.

Nombre	**Seleccionar personaje**
Actor	Jugador
Precondición	El usuario se encuentra autenticado y tiene al menos un personaje creado
Escenario	1. Se muestra el menú del jugador. 2. El usuario pulsa "Seleccionar personaje" sobre uno de sus personajes. 3. Se muestra la pantalla "Juego" del personaje.
Alternativo	-
Resultado	-

Nombre	**Información del personaje**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Seleccionar personaje"
Escenario	1. El jugador pulsa en "Personaje". 2. Se muestra el panel "Información del personaje" con los detalles del mismo: valores de salud, valores de habilidades, inventario.
Alternativo	-
Resultado	-

Nombre	**Información del entorno**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Seleccionar personaje"
Escenario	1. El jugador pulsa en "Entorno". 2. Se muestra el panel "Información del entorno" con los detalles del mismo: recursos disponibles, caminos de salida, personajes y objetos presentes, y trabajos comenzados.
Alternativo	-
Resultado	-

Nombre	**Información de actividad**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Seleccionar personaje"
Escenario	1. El jugador pulsa en "Actividad". 2. Se muestra el panel "Información de actividad" con los últimos eventos ocurridos: acciones realizadas por otros personajes, resultado de los trabajos, cambios en la salud. 3. Se muestra el listado de trabajos comenzados por cualquier personaje en la actual localización.
Alternativo	-
Resultado	-

Nombre	**Mover un elemento**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Seleccionar personaje"
Escenario	1. El jugador pulsa "Dar" sobre un elemento de su inventario. 2. Se muestra una ventana emergente solicitando el personaje receptor y la cantidad. 3. El jugador indica el personaje receptor y pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	1a. El jugador pulsa "Soltar" sobre un elemento de su inventario. 1a.1. Se muestra una ventana emergente solicitando la cantidad. 1a.2. El jugador pulsa "Aceptar". 1a.3. Vuelve a 4. 2a. El jugador pulsa "Coger" sobre un elemento del entorno. 2a.1. Se muestra una ventana emergente solicitando la cantidad. 2a.2. Vuelve a 4.
Resultado	La nueva acción queda registrada

Nombre	**Usar elemento**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información del personaje"
Escenario	1. El jugador pulsa "Usar" sobre un elemento de su inventario de tipo alimento. 2. Se muestra una ventana emergente de confirmación. 3. El jugador pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	1a. El jugador pulsa "Usar" sobre un elemento de tipo nota. 1a.1. Se muestra una ventana emergente con el contenido de la nota. 1a.2. El jugador pulsa "Aceptar". 1a.3. Vuelve a 5.
Resultado	La nueva acción queda registrada

Nombre	**Partir hacia camino**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información del entorno"
Escenario	1. El jugador pulsa "Tomar camino" sobre uno de los caminos. 2. Se muestra una ventana emergente de confirmación. 3. El jugador pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	-
Resultado	La nueva acción queda registrada

Nombre	**Extraer recurso**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información del entorno"
Escenario	1. El jugador pulsa "Extraer" sobre uno de los recursos. 2. Se muestra una ventana emergente de confirmación. 3. El jugador pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	-
Resultado	La nueva acción queda registrada

Nombre	**Susurrar**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información del entorno"
Escenario	1. El jugador pulsa "Susurrar" sobre uno de los personajes. 2. Se muestra una ventana emergente solicitando el mensaje. 3. El jugador introduce el mensaje y pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	-
Resultado	La nueva acción queda registrada

Nombre	**Atacar**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información del entorno"
Escenario	1. El jugador pulsa "Atacar" sobre uno de los personajes. 2. Se muestra una ventana emergente solicitando el arma a utilizar. 3. El jugador indica el arma y pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	-
Resultado	La nueva acción queda registrada

Nombre	**Hablar**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información del entorno"
Escenario	1. El jugador escribe un mensaje en la casilla y pulsa "Habla a todos". 2. Se registra la acción. 3. Se muestra un mensaje de éxito.
Alternativo	-
Resultado	La nueva acción queda registrada

Nombre	**Realizar trabajo**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información de actividad"
Escenario	1. El jugador pulsa "Crear trabajo". 2. Se muestra una ventana emergente solicitando los materiales necesarios. 3. El jugador indica los materiales y pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	1a. El jugador pulsa "Participar" sobre uno de los trabajos iniciados. 1a.1. Vuelve a 4. 2a. El jugador pulsa "Dejar" sobre el trabajo en el que está participando. 2a.1. Vuelve a 4.
Resultado	La nueva acción queda registrada

Nombre	**Escribir nota**
Actor	Jugador
Precondición	Se ha realizado el c.u. "Información de actividad"
Escenario	1. El usuario pulsa "Crear nota". 2. Se muestra una ventana emergente solicitando el mensaje. 3. El jugador indica el mensaje y pulsa "Aceptar". 4. Se registra la acción. 5. Se muestra un mensaje de éxito.
Alternativo	-
Resultado	La nueva acción queda registrada

6.6. Motor del sistema

Nombre	**Ejecutar acciones rápidas**
Actor	Reloj
Precondición	Existe al menos una acción en la cola de acciones rápidas
Escenario	1. Se leen los datos de la primera acción: jugador que la realiza, localización, tipo de acción y datos particulares de la misma. 2. Se actualizan los datos de personajes y elementos afectados. 3. Se escribe una descripción textual del evento para mostrar a los personajes que presencian la acción. 4. Se elimina la acción procesada de la cola.
Alternativo	-
Resultado	La acción se procesa y se borra del registro

Nombre	**Actualizar acciones lentas**
Actor	Reloj
Precondición	Ha pasado un determinado periodo de tiempo desde que se disparó el último turno
Escenario	1. Se incrementa el contador de turnos. 2. Para cada acción en la cola de acciones lentas: 2.1. Se leen los datos de la primera acción lenta o trabajo: jugador que la realiza, localización, tipo de acción, duración total, progreso actual. 2.2. Se actualiza el progreso de la acción o trabajo.
Alternativo	2a. No existen acciones en la cola y termina el c.u. 2.2a. Se actualiza el progreso de la acción y llega a su fin. 2.2a.1. Se escribe una descripción textual del evento para mostrar a los personajes que presencian la acción. 2.2a.2. Se elimina la acción procesada de la cola.
Resultado	Se incrementa el turno y se actualizan las acciones del turno anterior

7. Entidades

Previo al diseño, identificaremos las principales entidades del sistema, los atributos de cada una y las relaciones entre ellas. Para simplificar no mostraremos los atributos identificadores ni incluimos todavía los métodos. El diagrama de clases general se muestra en las fig. 7 en la página siguiente.

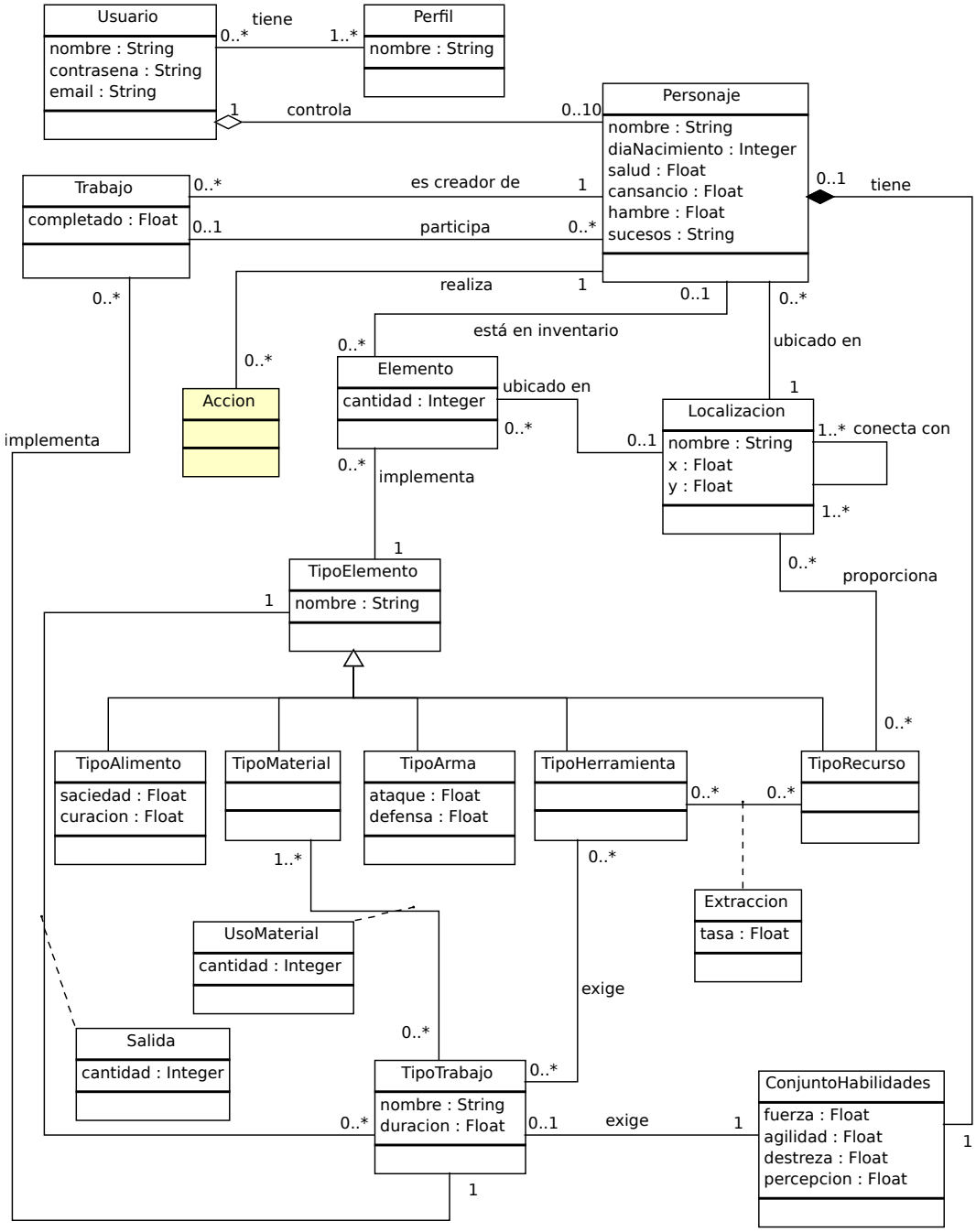


Figura 7: Diagrama de clases general.

La entidad “Accion” se detalla con más precisión en la fig. 8 en la página siguiente, ya que existen muchos subtipos de acción con diversos matices cada uno. En la figura se han resaltado con un color distinto al blanco las clases repetidas en el diagrama anterior.

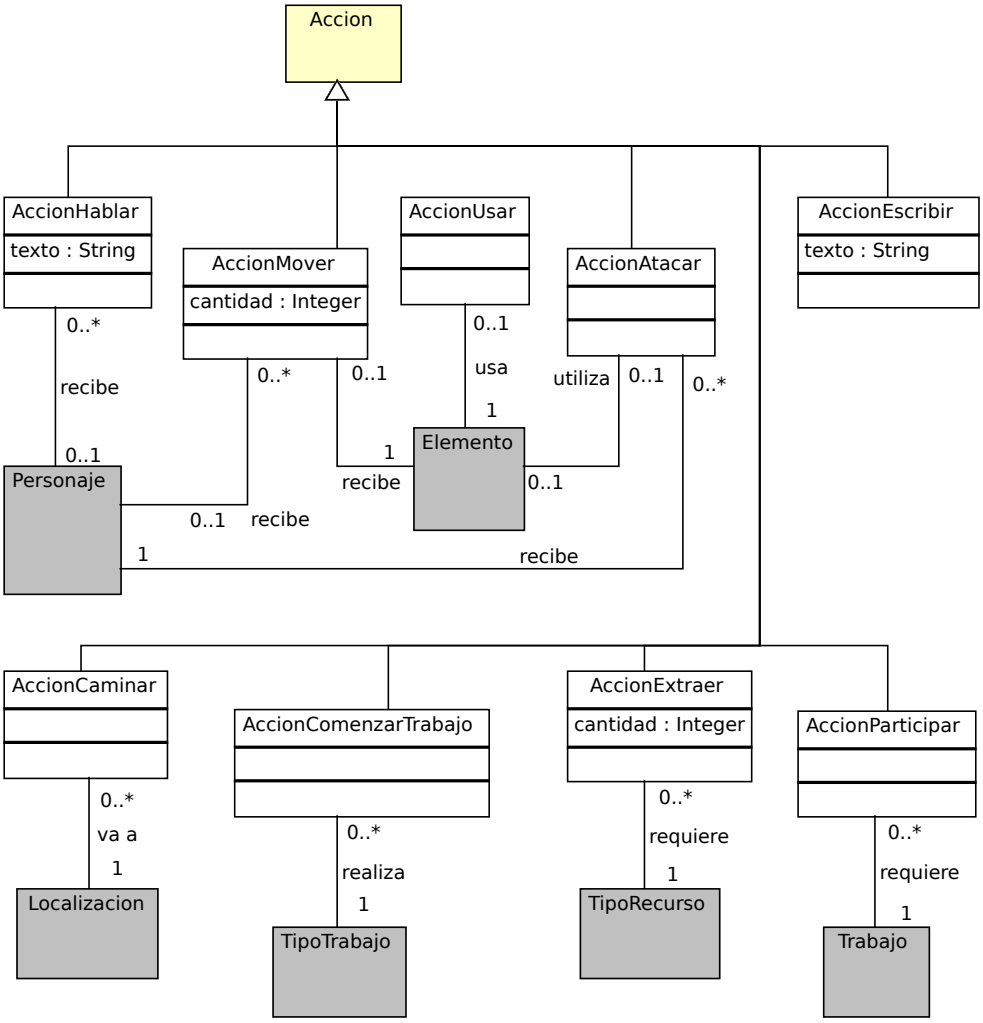


Figura 8: Diagrama de clases. Detalle de la clase "Accion".

8. Interfaz con el usuario

Los casos de uso descritos hacen referencia a determinadas páginas que tendremos que realizar en la fase de implementación. No obstante, mostraremos un prototipo de la interfaz con el usuario que sirva como complemento a estas descripciones y faciliten la fase de implementación.

Figura 9: Pantalla "Autenticar"

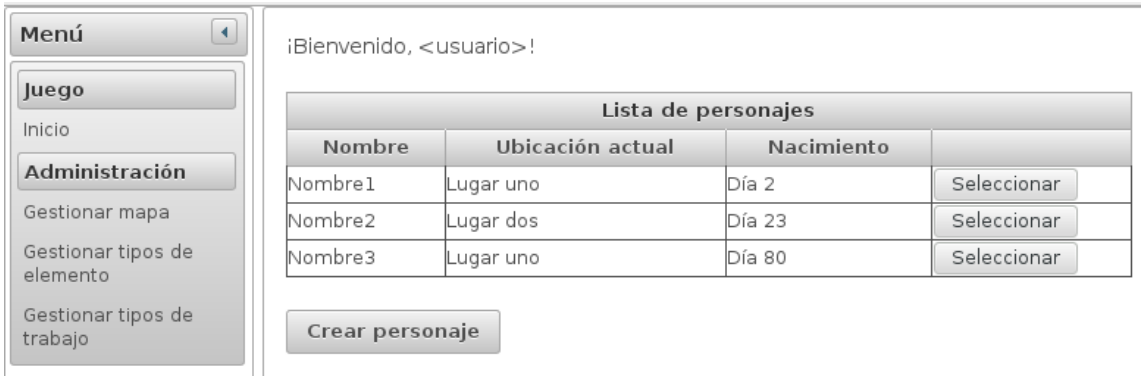


Figura 10: Pantalla "Inicio"

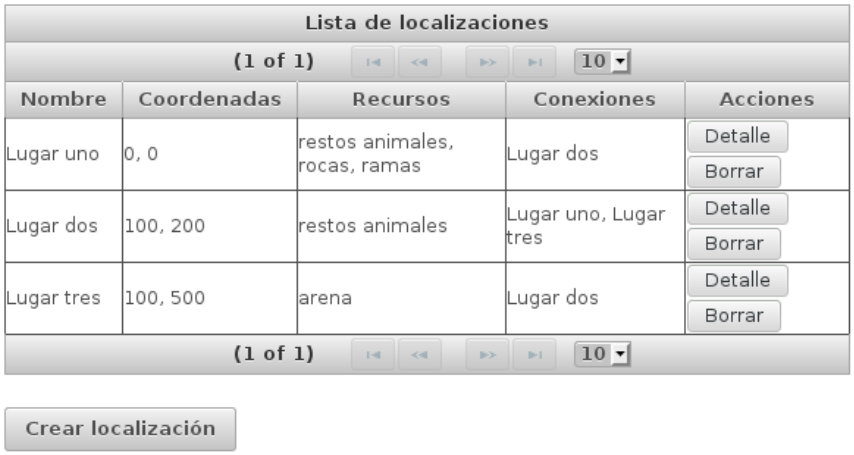


Figura 11: Pantalla "Gestión del mapa"

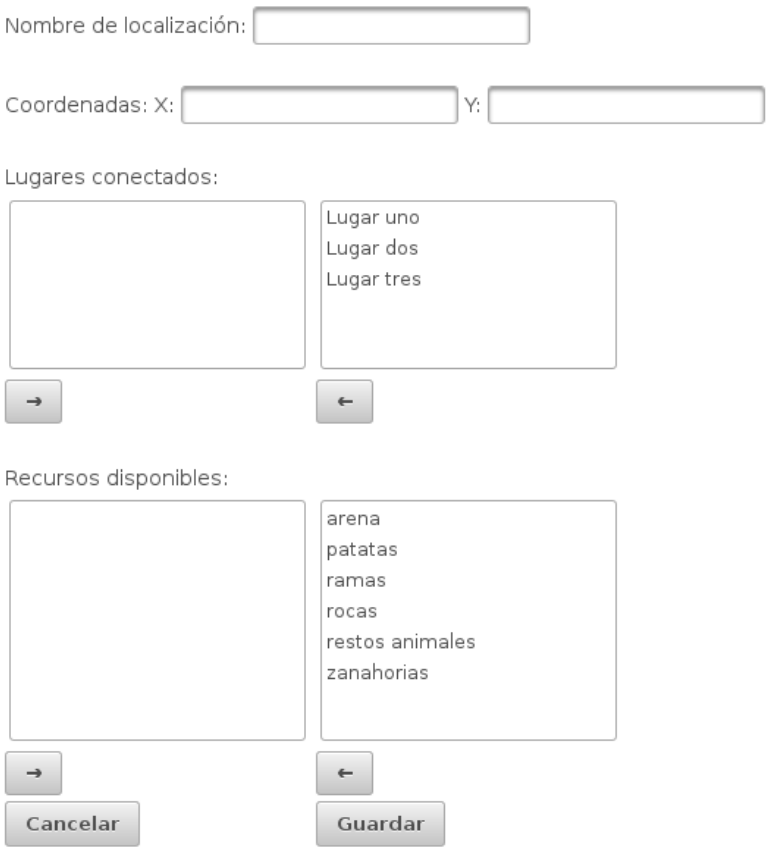


Figura 12: Pantalla "Detalle de localización"

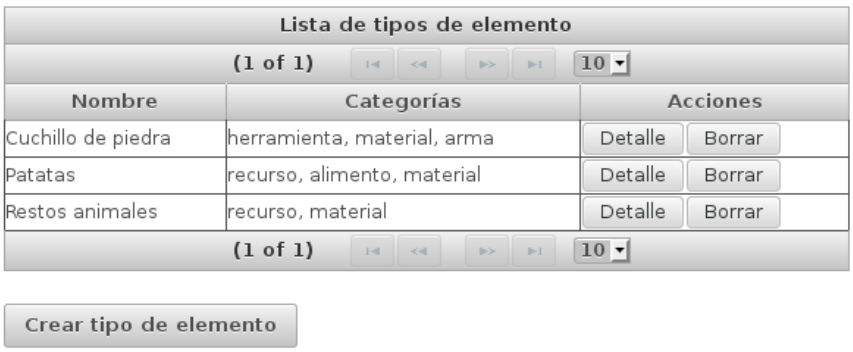


Figura 13: Pantalla "Gestión de tipos de elemento"

Figura 14: Pantalla "Detalle de tipo de elemento"

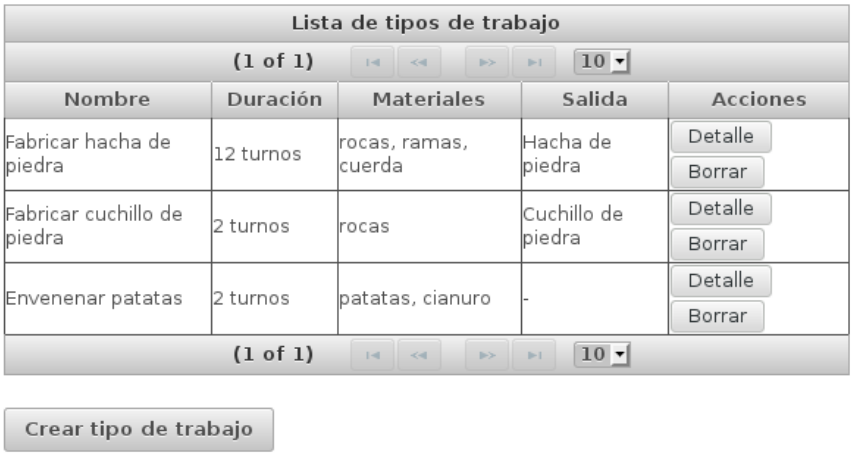


Figura 15: Pantalla "Gestión de tipos de trabajo"

Figura 16: Pantalla "Detalle de tipo de trabajo"

Figura 17: Pantalla "Juego"

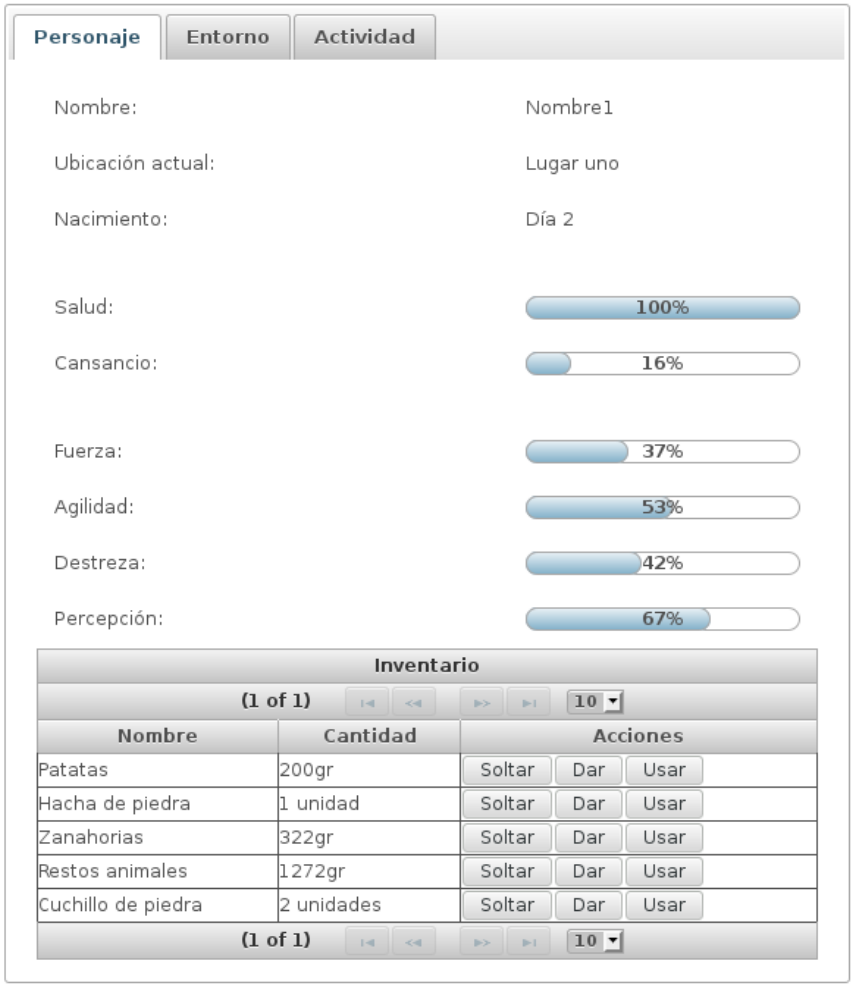


Figura 18: Panel "Información del personaje"

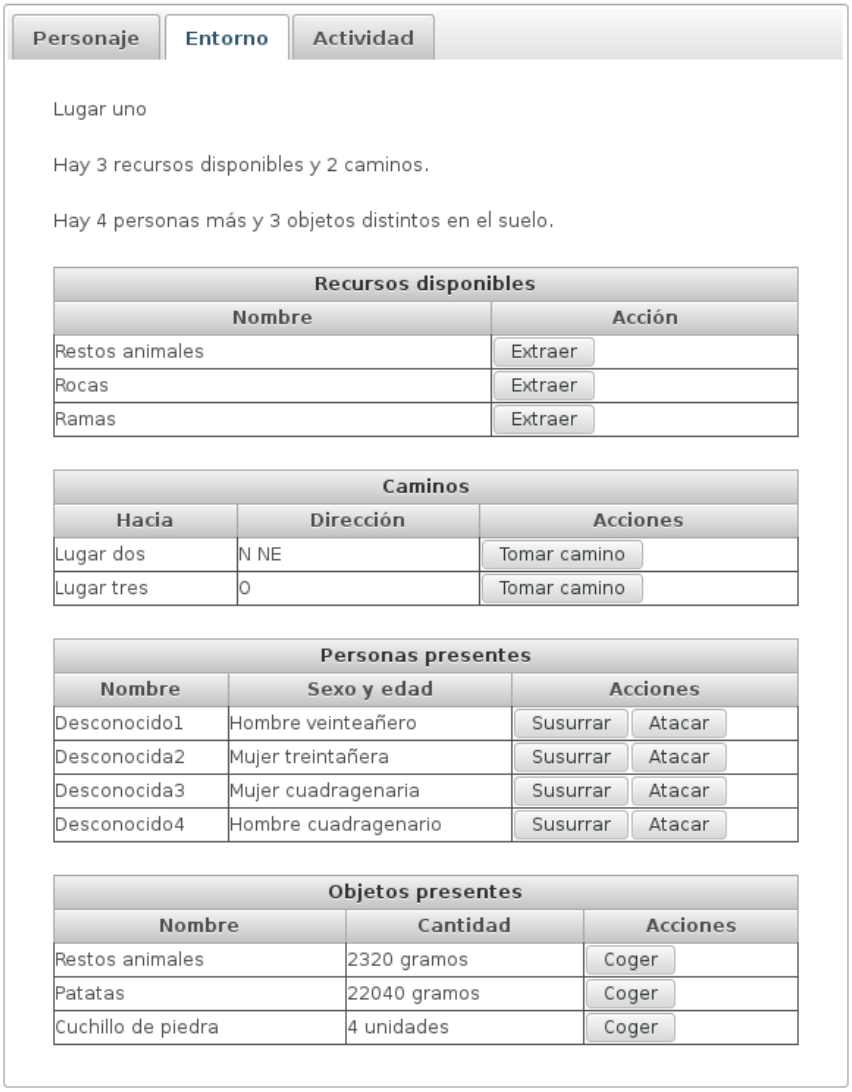


Figura 19: Panel “Información del entorno”

Figura 20: Panel "Información de actividad"

9. Posibles mejoras

Las posibilidades de un juego de estas características prácticamente solo están limitadas por nuestra imaginación y por el tiempo y esfuerzo que podamos aportar. No obstante, existen algunas mejoras que serían —relativamente— fáciles de implementar, pero que hemos decidido omitir para contener el alcance de este proyecto. Algunas de ellas serían:

- Gestión de los usuarios —por parte del administrador— dentro de la propia aplicación.
- Que los personajes mueran cuando su salud desciende hasta un 0 %.
- Posibilidad de hacer notas editables y no editables, así como de leer las que estén en el suelo de la localización.
- Posibilidad de aportar los materiales necesarios paulatinamente y después de crear el trabajo, en lugar de al crearlo.
- Posibilidad de elegir la cantidad de materiales (proporcionalmente a los establecidos) para el trabajo, en lugar de ser cantidades fijas.
- Posibilidad de elegir la cantidad que se desea extraer o bien el número de turnos que se desean dedicar para la extracción de recursos.
- Aprendizaje de los personajes: las habilidades aumentan conforme a su uso.
- Creación de edificios: localizaciones anidadas dentro de otra localización.

- Recursos naturales limitados o de crecimiento dinámico.
- Y un largo etcétera...

Parte IV
Diseño

Durante la fase de diseño describiremos la arquitectura del sistema y mostraremos cómo se debe codificar el funcionamiento del mismo mediante los diagramas de secuencia.

10. Arquitectura lógica

La fase de análisis dividió el proyecto en módulos transversales según las funcionalidades que fuera a realizar cada uno. Ahora es el momento de diseñar la arquitectura vertical que abarca desde la interfaz con el usuario hasta el registro y lectura de toda la información en la base de datos. Para ello dividiremos el proyecto en las tres capas clásicas: presentación, lógica de negocio y persistencia. Detallaremos cada una de estas capas en los siguientes apartados.

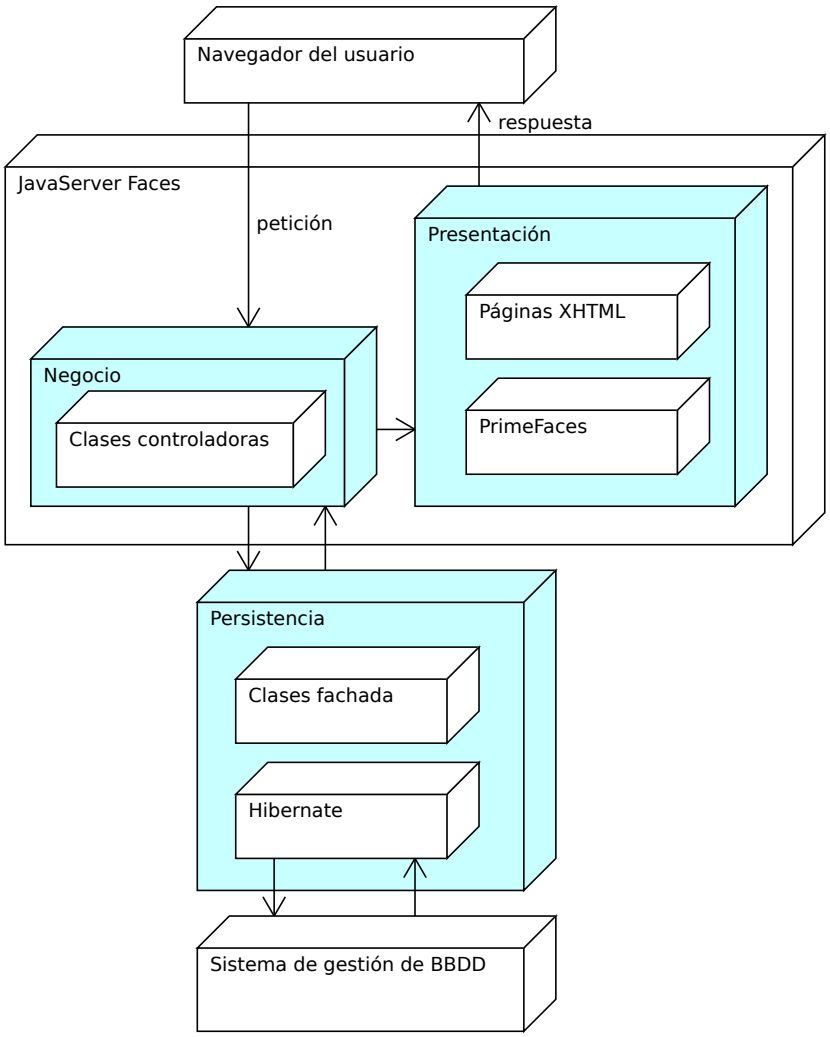


Figura 21: Esquema de la arquitectura lógica del sistema.

10.1. Presentación

La capa de presentación se encargará de implementar la interfaz con el usuario. De cara a este, la interfaz está compuesta por las pantallas en formato HTML que serán accesibles mediante su navegador web.

Como no se trata de una simple página web estática, el servidor de aplicaciones debe realizar una serie de operaciones que le permitan generar la información personalizada, exacta y en el formato correcto. También deberá recoger las instrucciones e información que el usuario desea introducir en el sistema y mostrarle los mensajes pertinentes.

La capa software encargada de esto, como ya hemos dicho, es la capa de persistencia. Y en nuestro caso diremos que está formada por:

JavaServer Faces (JSF) es una de las tecnologías específicas para implementar el interfaz web en Java EE. Alrededor de ella giran el resto de los componentes aquí descritos. Utilizaremos la versión 2.1.

PrimeFaces es simplemente una extensión para JSF que incluye más componentes, facilitando el trabajo con estos y dotándolos de mayores posibilidades.

Las páginas XHTML conforman la vista propiamente dicha. Contendrán componentes HTML, pero también se indicará cómo y dónde debe ir cada componente JSF. Es necesario complementar lo anterior con instrucciones programáticas en lenguage EL (*Expression Language*) intercaladas en cada documento.

Los ficheros de mensajes guardan todas las etiquetas y textos estáticos que el usuario podrá ver por pantalla, de forma que sean fácilmente traducibles a otros idiomas y modificables sin tener que manipular el código fuente.

El esfuerzo de desarrollo de la capa de presentación irá primordialmente a realizar las páginas XHTML. Existirá, aproximadamente, una página por cada una de las pantallas que resultaron durante la fase de análisis.

10.2. Negocio

La capa de negocio se encargará de interpretar qué quiere el usuario —según las instrucciones que lleguen de la capa superior—, realizar las operaciones lógicas necesarias, y mandar el resultado de estas operaciones de vuelta a la capa de presentación. Para ello también deberá comunicarse con la capa inferior (persistencia) con el objetivo de leer o guardar datos.

Las clases controladoras (*controller*) serán el componente principal de esta capa. Estarán implementadas como *managed beans* y dentro de ellas codificaremos la lógica de negocio. Existirá una clase controladora por cada módulo o grupo de funcionalidades bien definida.

Clases controladoras

Crearemos siete clases controladoras distintas. Detallaremos la funcionalidad de la que es responsable de cada una de ellas sen las siguientes tablas. También mostraremos los principales atributos y métodos, aunque no añadiremos los parámetros a estos últimos.

LoginControl	
Función	Controlará la autenticación del usuario al sistema, así como el acceso a los distintos menús según su perfil. También manejará la sesión según el personaje que elija el jugador en cada momento.
Atributos	usuarioFacade: UsuarioFacade perfilFacade: PerfilFacade personajeFacade: PersonajeFacade
Operaciones	entra() creaPersonaje()

LocalizacionControl	
Función	Controlará el módulo de gestión de localizaciones.
Atributos	localizacionFacade: LocalizacionFacade tipoRecursoFacade: TipoRecursoFacade tipoHerramientaFacade: TipoHerramientaFacade extraccionFacade: ExtraccionFacade
Operaciones	creaLocalizacion() detalleLocalizacion() borraLocalizacion() guardaLocalizacion() anadeCamino() anadeTipoRecurso()

TipoElementoControl	
Función	Controlará el módulo de gestión de tipos de elemento.
Atributos	tipoElementoFacade: TipoElementoFacade tipoAlimentoFacade: TipoAlimentoFacade tipoMaterialFacade: TipoMaterialFacade tipoArmaFacade: TipoArmaFacade tipoRecursoFacade: TipoRecursoFacade tipoHerramientaFacade: TipoHerramientaFacade extraccionFacade: ExtraccionFacade
Operaciones	crearTipoElemento() detalleTipoElemento() borrarTipoElemento() guardarTipoElemento() anadirExtraccion()

TipoTrabajoControl	
Función	Controlará el módulo de gestión de tipos de trabajo.
Atributos	tipoTrabajoFacade: TipoTrabajoFacade tipoMaterialFacade: TipoMaterialFacade tipoElementoFacade: TipoElementoFacade usoMaterialFacade: UsoMaterialFacade conjuntoHabilidadesFacade: ConjuntoHabilidadesFacade tipoHerramientaFacade: TipoHerramientaFacade
Operaciones	creaTipoTrabajo() detalleTipoTrabajo() borraTipoTrabajo() guardaTipoTrabajo() anadeUsoMaterial() anadeTipoHerramienta() anadeTipoElemento()

PersonajeControl	
Función	Controlará, dentro del módulo de gestión del personaje, la creación y selección e información del mismo y de su entorno.
Atributos	personajeFacade: PersonajeFacade conjuntoHabilidadesFacade: ConjuntoHabilidadesFacade tipoElementoFacade: TipoElementoFacade elementoFacade: ElementoFacade localizacionFacade: LocalizacionFacade tipoRecursoFacade: TipoRecursoFacade notaFacade: NotaFacade
Operaciones	seleccionaPersonaje() muestraLocalizacion() muestraRecursos() muestraElementos() muestraInventario() muestraPersonajes() muestraSucesos() muestraTrabajos()

ActividadControl	
Función	Controlará, dentro del módulo de gestión del personaje, la creación de actividades por parte del mismo.
Atributos	actividadFacade: ActividadFacade tipoTrabajoFacade: TipoTrabajoFacade personajeFacade: PersonajeFacade trabajoFacade: TrabajoFacade notaFacade: NotaFacade usoMaterialFacade: UsoMaterialFacade accionFacade: AccionFacade
Operaciones	sueltaElemento() cogeElemento() daElemento() usaElemento() extrae() tomaCamino() ataca() susurra() habla() creaNota() creaTrabajo() participaTrabajo() dejaTrabajo()

SistemaControl	
Función	Controlará el despacho de las actividades por parte del reloj del sistema.
Atributos	accionFacade: AccionFacade personajeFacade: PersonajeFacade elementoFacade: ElementoFacade localizacionFacade: LocalizacionFacade tipoTrabajoFacade: TipoTrabajoFacade tipoRecursoFacade: TipoRecursoFacade trabajoFacade: TrabajoFacade
Operaciones	procesaAccion() pasaTurno()

10.3. Persistencia

Para realizar la persistencia de los datos vamos a utilizar el estándar Java Persistence API (JPA) en su versión 2. Esta capa debe permitirnos leer, actualizar, crear y borrar cualquier entidad en la base de datos de forma transparente. Debería permitirnos escoger cualquiera de los principales sistemas de gestión de base de datos sin tener que cambiar la capa propiamente dicha; tal vez, algún fichero de configuración. Aunque nuestra elección concreta será PostgreSQL en su versión 9, ya que en un futuro puede ser reco-

mendable ampliar sus capacidades cartográficas —permitiendo una mejor gestión de las localizaciones— mediante el módulo PostGIS.

Para nuestro proyecto utilizaremos las siguientes bibliotecas y componentes:

Hibernate es la implementación concreta de JPA 2 que usaremos.

Las clases de tipo fachada (*facade*) implementarán las operaciones básicas de lectura y escritura en base de datos haciendo uso de la interfaz EntityManager (cuya instancia será proporcionada por el contexto de persistencia) específica de cada entidad. Serán inyectadas como Enterprise JavaBeans (EJB) en las clases controladoras.

Las clases de tipo entidad (*entity*) contendrán de forma muy simplificada los atributos y los métodos básicos de acceso a estos. Aunque las incluimos como parte de la capa de persistencia, las entidades son utilizadas en todas las capas como vehículo de los datos a través de ellas.

Clases fachada

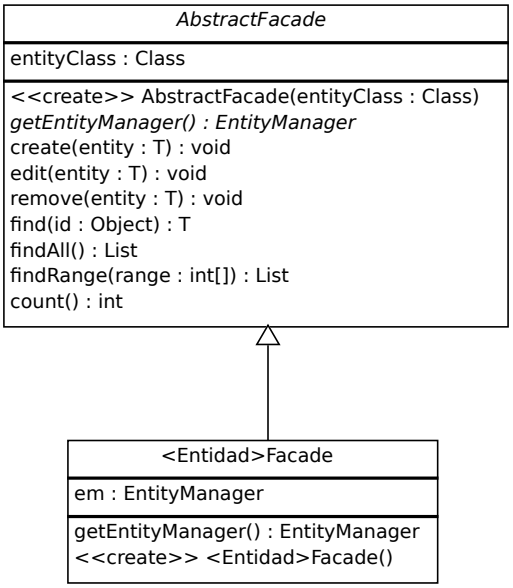


Figura 22: Diagrama de las clases de tipo fachada.

A la hora de implementar la persistencia, merece especial atención la clase AbstractFacade. Será una clase abstracta que tendrá que especializarse en cada una de las demás clases fachada. Las operaciones básicas que proporciona esta clase están implementadas de forma paramétrica, por lo que no es necesario reemplazarlas por especializaciones. Solamente deberemos implementar las operaciones más específicas de cada entidad. En la fig. 22 se muestra el diagrama de clases correspondiente. Nótese que —para simplificar y evitar repeticiones innecesarias— las clases hijas se han representado como una sola clase genérica.

A continuación se describen los métodos principales:

AbstractFacade	
getEntityManager()	Método abstracto que devolverá la instancia EntityManager de la clase fachada que estemos tratando en un momento dado, por lo que debe ser implementado en las clases hijas.
create(entity: T)	Guarda la nueva entidad que se pasa como parámetro y la hace persistente, de forma que posteriores cambios serán seguidos y actualizados automáticamente en base de datos.
edit(entity: T)	Actualiza la fila correspondiente de base de datos con los valores concretos de la entidad que se pasa como parámetro.
remove(entity: T)	Borra la fila correspondiente a la entidad.
find(id: Object)	Busca una entidad por su identificador, y la devuelve.
findAll()	Devuelve una lista de todas las entidades.
findRange(range: int[])	Devuelve una lista acotada de las entidades. Nos será útil para realizar la paginación en listados largos.
count()	Devuelve un entero con el número total de filas existentes.

Clases entidad

El diseño de las clases de tipo entidad para la persistencia respeta el diagrama de clases y los requisitos generados durante la fase de análisis. Pero trataremos de adaptar las múltiples subclases que fueron detectadas a las tecnologías de desarrollo, intentando mantener un compromismo entre eficiencia y facilidad de uso y reutilización de componentes.

Para realizar este diseño, se ha optado por el uso de *annotations* de JPA 2. Esto supone que describiremos las relaciones y otros detalles de las tablas dentro del propio código java de la clase. A continuación mostraremos el esquema de estas clases; y en el Anexo I dejaremos el código SQL autogenerado por JPA 2.

Prescindiremos de las operaciones de cada clase, ya que solo tendremos los *setters* y *getters* de los atributos mostrados, y las operaciones básicas hashCode(), equals() y toString().

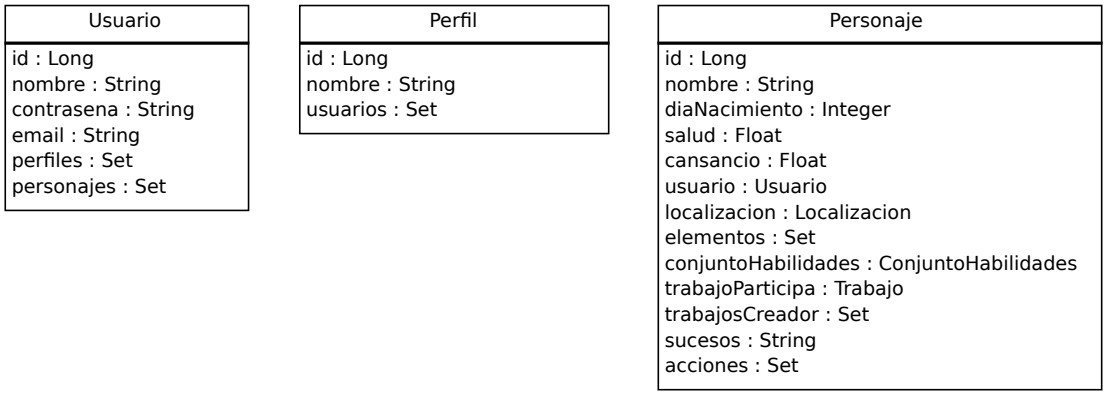


Figura 23: Clases Usuario, Perfil y Personaje.

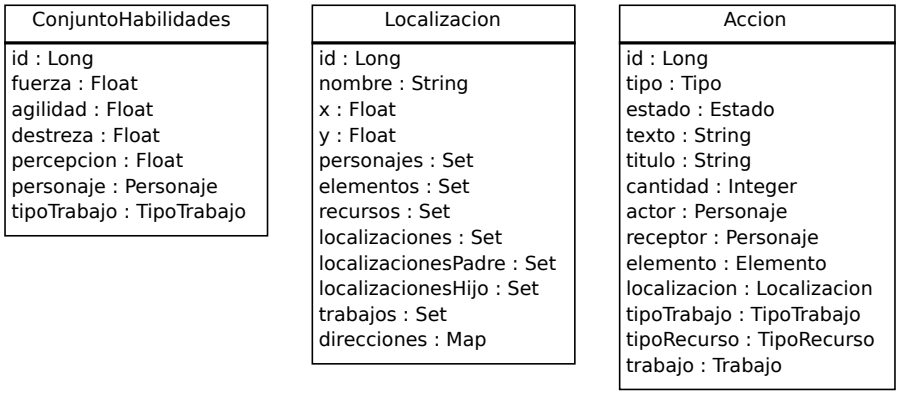


Figura 24: Clases ConjuntoHabilidades, Localizacion y Accion.

Figura 25: Clases Extraccion, Trabajo y UsoMaterial.

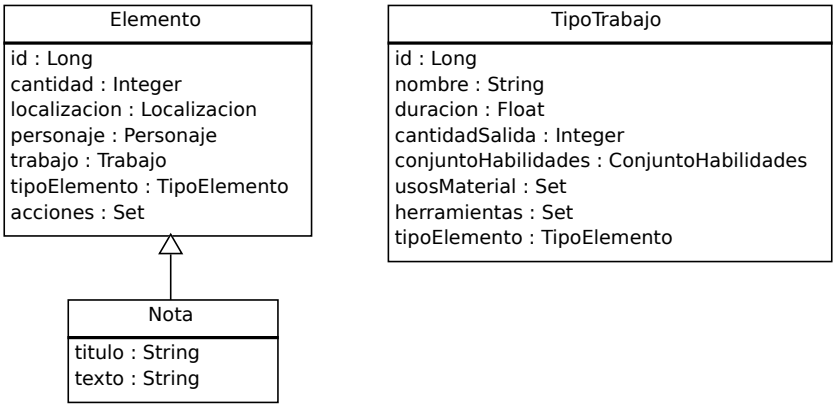


Figura 26: Clases Elemento, Nota y TipoTrabajo.

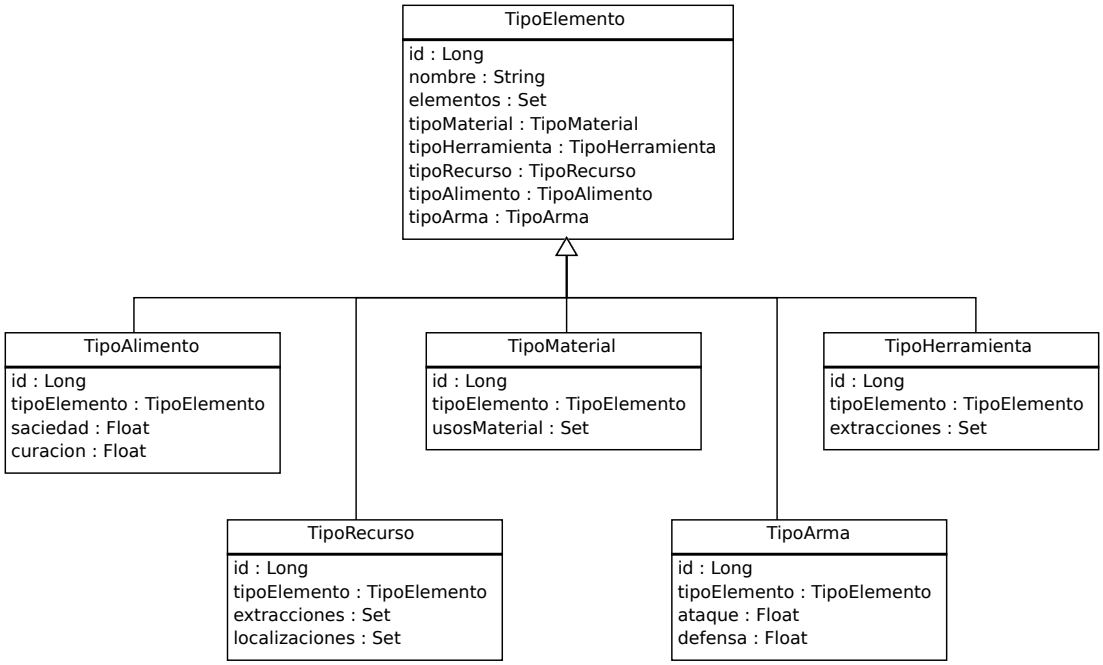


Figura 27: Clase TipoElemento y sus subclases.

11. Diagramas de secuencia

Ilustraremos la utilización de las capas y los objetos de cada una de ellas mediante los diagramas de secuencia. Para simplificar, y por razones de espacio, solo mostraremos los diagramas de secuencia que realizan los casos de uso más importantes o representativos. Por esto, dejaremos de lado los módulos de gestión de localizaciones, tipos de elemento y tipos de trabajo —ya que se tratan de *cruds* clásicos—; y nos centraremos en la gestión del personaje y el motor del sistema.

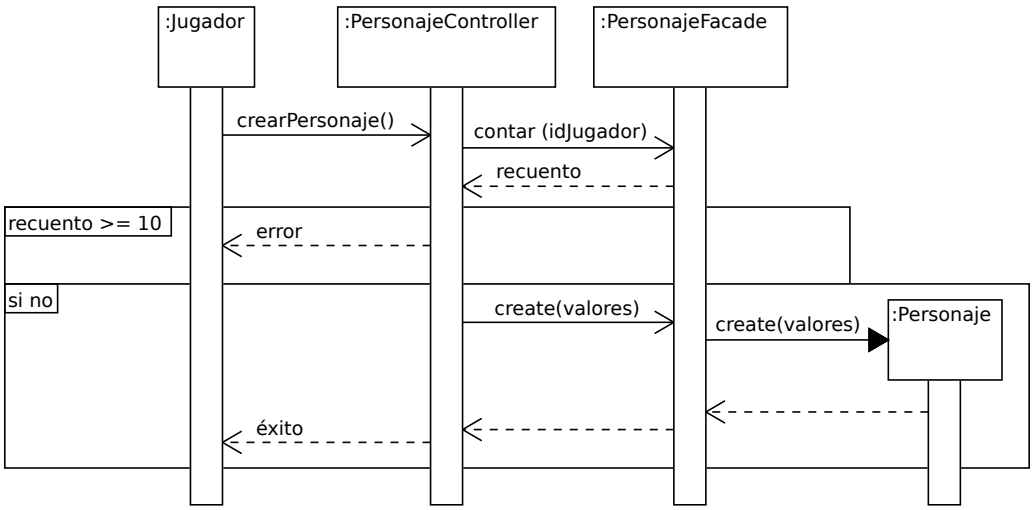


Figura 28: Diagrama de secuencia para "Crear personaje".

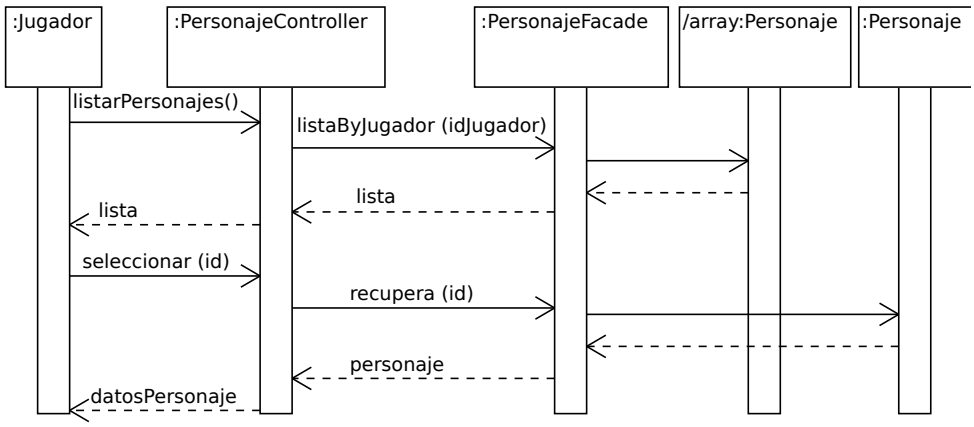


Figura 29: Diagrama de secuencia para "Seleccionar personaje".

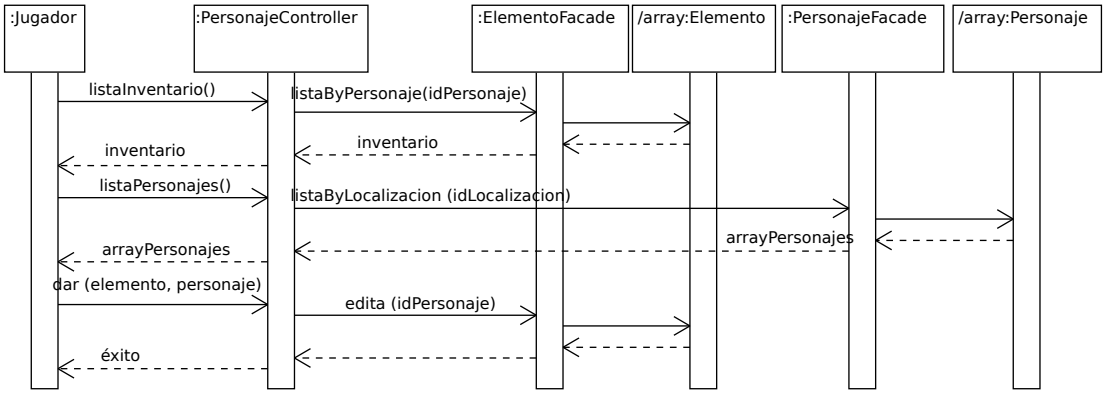


Figura 30: Diagrama de secuencia para "Mover un elemento".

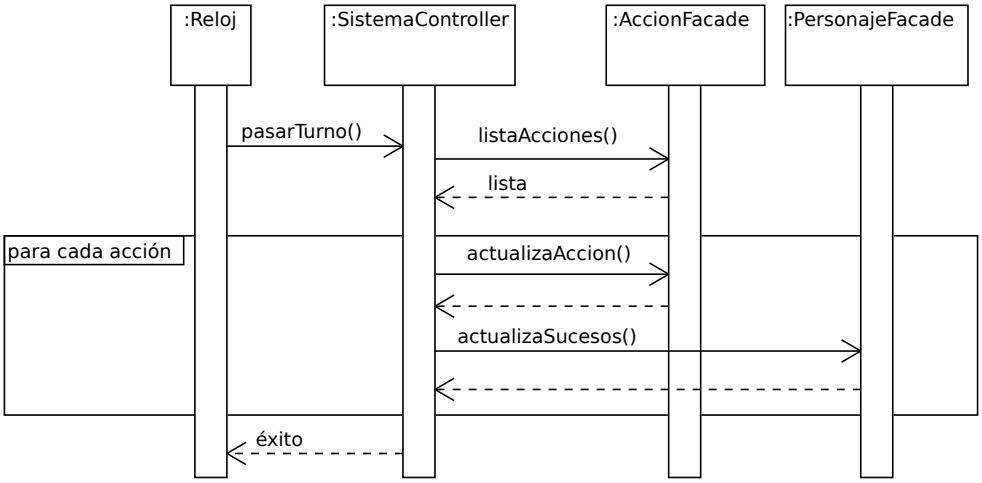


Figura 31: Diagrama de secuencia para “Actualizar acciones lentas”.

Parte V
Implementación

Por muy bueno que sea un diseño, en la práctica casi siempre nos encontraremos —a la hora de implementar— con pequeños detalles que previamente no conocíamos; sobre todo cuando no somos expertos en todas las tecnologías que se utilizan. El carácter didáctico de este trabajo ha hecho que durante la fase de implementación adquiramos bastantes conocimientos, experimentando y encontrando dificultades "imprevistas", aunque también sorpresas positivas.

Por todo esto, es importante que en las fases anteriores jugáramos con cierto "margen de error" para la fase de implementación. Aunque en general hemos conseguido nuestros objetivos, hemos encontrado algunos problemas y lecciones aprendidas a tener en cuenta en futuras iteraciones que realicemos a esta primera versión del software.

12. Carencias detectadas

Aunque las horas invertidas en el aprendizaje y la resolución de problemas han sido muchas, resumimos algunas de las carencias que hemos encontrado en la fase previa a la entrega del producto software. Esto es importante para futuras versiones del software que pretendamos desarrollar. Por ejemplo, tendremos que revisar:

- Las estructuras de datos relativas a los casos de uso "extraer un recurso" y "partir hacia un camino". Se ha utilizado la clase Trabajo para implementar estas acciones, incluyendo un par de nuevos atributos con los que no contamos en la fase de diseño. Posiblemente la herencia hubiera sido un enfoque mejor para nuestros objetivos.
- Actualización de los sucesos: en la vista de juego, debemos procurar que el usuario siempre disponga de datos actualizados tras cada llamada al servidor. Por otra parte, esto supone un compromiso con respecto a la velocidad del juego. Durante las pruebas realizadas, nos encontramos con que los sucesos tras un cambio de turno no son actualizados en el acto.
- Documentación: aunque la estructura está claramente definida en la fase de diseño, y los nombres de los métodos y atributos de clase son lo más descriptivos posible, es necesario mejorar este aspecto con más comentarios dentro del código fuente.
- Lógica de negocio: no se han implementado completamente los métodos que calculan las tasas de extracción o el avance de los trabajos según los conjuntos de habilidades.

13. Mejoras añadidas

Como decíamos, el carácter didáctico de este ejercicio nos ha permitido explorar mientras implementábamos. Es por ello que no hemos desaprovechado la oportunidad de mejorar ciertos aspectos del producto cuando el esfuerzo empleado iba a ser pequeño; y la mejora, notable:

- En cada localización se indica la dirección a la que llevan los caminos hacia otras localizaciones, refiriendo 16 posibles puntos cardinales distintos. Por ejemplo: Norte, Nornoroeste, Noroeste, Oestenoroeste, Oeste...
- Identificador visual para cada personaje. No ponemos como restricción que los nombres sean únicos, dejando así total libertad a los jugadores. Como contrapartida, se pueden dar posibles confusiones dentro del juego que no serían realistas. Este problema lo solucionaremos mostrando identificadores visuales únicos para cada personaje, generados a través de un hash y recuperando la imagen correspondiente mediante la herramienta web Gravatar.
- Las notas tienen título además de cuerpo de texto. Así son fácilmente identificables en el inventario o el listado de objetos en el suelo.
- Se pueden leer las notas que hay en el suelo. Esto permite a los personajes dejar notas "públicas" que todos los presentes puedan leer en cualquier momento.

Parte VI
Conclusiones

El uso de tecnologías tan potentes y recientes como JSF 2, JPA 2 y EJB 3.1 hacen que, por un lado, la curva de aprendizaje durante el desarrollo haya sido dura de superar. Por otra parte, nos ha permitido elaborar una aplicación de una cierta complejidad mediante poco código fuente, lo que hace que sea fácilmente mantenible y escalable.

La depuración de errores tampoco ha sido demasiado complicada, debiéndose muchas veces al desconocimiento de algún aspecto de estas tecnologías y que, por tanto, se han podido resolver consultando la documentación oficial o foros de resolución de dudas.

En cuanto a lo aprendido, el resultado ha sido muy satisfactorio desde un punto de vista personal. JavaEE 6 demuestra ser una tecnología muy asentada, que va mejorando aún más su grado de madurez y que nos da alas a los desarrolladores para poder plasmar casi cualquier idea o proyecto que se nos pase por la cabeza.

Parte VII
Anexo I – Generación de las tablas SQL

A continuación se muestran las sentencias SQL básicas para generar las tablas necesarias en la base de datos. Hay que recordar que, gracias a la forma en la que hemos usado JPA 2, estas tablas se generan automáticamente al arrancar el servidor de aplicaciones y desplegar nuestro software.

```
CREATE TABLE accion (
    id bigint NOT NULL,
    cantidad integer,
    creado timestamp without time zone,
    estado integer NOT NULL,
    texto character varying(32768),
    tipo integer NOT NULL,
    titulo character varying(128),
    actor_id bigint NOT NULL,
    elemento_id bigint,
    localizacion_id bigint,
    receptor_id bigint,
    tiporecurso_tipoelemento_id bigint,
    tipotrabajo_id bigint,
    trabajo_id bigint,
    CONSTRAINT accion_cantidad_check CHECK (((cantidad >= 1) AND (
        cantidad <= 9223372036854775807::bigint)))
);

CREATE TABLE conjuntohabilidades (
    id bigint NOT NULL,
    agilidad real NOT NULL,
    destreza real NOT NULL,
    fuerza real NOT NULL,
    percepcion real NOT NULL
);

CREATE TABLE elemento (
    id bigint NOT NULL,
    cantidad integer NOT NULL,
    localizacion_id bigint,
    personaje_id bigint,
    tipoelemento_id bigint NOT NULL,
    trabajo_id bigint
);

CREATE TABLE extraccion (
    id bigint NOT NULL,
    tasa real NOT NULL,
    tipoherramienta_id bigint,
    tiporecurso_id bigint NOT NULL
);
```

```
CREATE TABLE localizacion (
    id bigint NOT NULL,
    nombre character varying(255) NOT NULL,
    x real NOT NULL,
    y real NOT NULL
);

CREATE TABLE localizacion_localizacion (
    localizacionespadre_id bigint NOT NULL,
    localizacioneshijo_id bigint NOT NULL
);

CREATE TABLE localizacion_tiporecurso (
    localizaciones_id bigint NOT NULL,
    recursos_tipoelemento_id bigint NOT NULL
);

CREATE TABLE nota (
    texto character varying(32768) NOT NULL,
    titulo character varying(128) NOT NULL,
    id bigint NOT NULL
);

CREATE TABLE perfil (
    id bigint NOT NULL,
    nombre character varying(32) NOT NULL
);

CREATE TABLE personaje (
    id bigint NOT NULL,
    cansancio real NOT NULL,
    dianacimiento integer NOT NULL,
    nombre character varying(255) NOT NULL,
    salud real NOT NULL,
    sucesos character varying(131072),
    conjuntohabilidades_id bigint NOT NULL,
    localizacion_id bigint NOT NULL,
    trabajoparticipa_id bigint,
    usuario_id bigint NOT NULL
);

CREATE TABLE sistema (
    id bigint NOT NULL,
    creado timestamp without time zone NOT NULL,
    estado integer NOT NULL,
    turno bigint NOT NULL,
    turnospordia smallint NOT NULL
);

CREATE TABLE tipoalimento (
    curacion real NOT NULL,
    saciedad real NOT NULL,
    tipoelemento_id bigint NOT NULL
);

CREATE TABLE tipoarma (
```

```
    ataque real NOT NULL,
    defensa real NOT NULL,
    tipoelemento_id bigint NOT NULL
);

CREATE TABLE tipoelemento (
    id bigint NOT NULL,
    nombre character varying(255) NOT NULL
);

CREATE TABLE tipoherramienta (
    tipoelemento_id bigint NOT NULL
);

CREATE TABLE tipomaterial (
    tipoelemento_id bigint NOT NULL
);

CREATE TABLE tiporecurso (
    tipoelemento_id bigint NOT NULL
);

CREATE TABLE tipotrabajo (
    id bigint NOT NULL,
    cantidadsalida integer NOT NULL,
    duracion real NOT NULL,
    nombre character varying(255) NOT NULL,
    conjuntohabilidades_id bigint NOT NULL,
    tipoelemento_id bigint NOT NULL
);

CREATE TABLE tipotrabajo_tipoherramienta (
    tipotrabajo_id bigint NOT NULL,
    herramientas_tipoelemento_id bigint NOT NULL
);

CREATE TABLE trabajo (
    id bigint NOT NULL,
    completado real NOT NULL,
    localizacion_id bigint NOT NULL,
    localizaciondestino_id bigint,
    personajecreador_id bigint NOT NULL,
    tiporecurso_tipoelemento_id bigint,
    tipotrabajo_id bigint NOT NULL
);

CREATE TABLE usomaterial (
    id bigint NOT NULL,
    cantidad integer NOT NULL,
    tipomaterial_id bigint NOT NULL,
    tipotrabajo_id bigint NOT NULL
);

CREATE TABLE usuario (
    id bigint NOT NULL,
    contrasena character varying(255) NOT NULL,
```

```
    email character varying(255) NOT NULL,
    nombre character varying(255) NOT NULL
);

CREATE TABLE usuario_perfil (
    usuarios_id bigint NOT NULL,
    perfiles_id bigint NOT NULL
);

ALTER TABLE ONLY accion
    ADD CONSTRAINT accion_pkey PRIMARY KEY (id);

ALTER TABLE ONLY conjuntohabilidades
    ADD CONSTRAINT conjuntohabilidades_pkey PRIMARY KEY (id);

ALTER TABLE ONLY elemento
    ADD CONSTRAINT elemento_pkey PRIMARY KEY (id);

ALTER TABLE ONLY extraccion
    ADD CONSTRAINT extraccion_pkey PRIMARY KEY (id);

ALTER TABLE ONLY extraccion
    ADD CONSTRAINT extraccion_tipoherramienta_id_tiporecurso_id_key
        UNIQUE (tipoherramienta_id, tiporecurso_id);

ALTER TABLE ONLY localizacion_localizacion
    ADD CONSTRAINT localizacion_localizacion_pkey PRIMARY KEY (
        localizacionespadre_id, localizacioneshijo_id);

ALTER TABLE ONLY localizacion
    ADD CONSTRAINT localizacion_nombre_key UNIQUE (nombre);

ALTER TABLE ONLY localizacion
    ADD CONSTRAINT localizacion_pkey PRIMARY KEY (id);

ALTER TABLE ONLY localizacion_tiporecurso
    ADD CONSTRAINT localizacion_tiporecurso_pkey PRIMARY KEY (
        localizaciones_id, recursos_tipoelemento_id);

ALTER TABLE ONLY nota
    ADD CONSTRAINT nota_pkey PRIMARY KEY (id);

ALTER TABLE ONLY perfil
    ADD CONSTRAINT perfil_nombre_key UNIQUE (nombre);

ALTER TABLE ONLY perfil
    ADD CONSTRAINT perfil_pkey PRIMARY KEY (id);

ALTER TABLE ONLY personaje
    ADD CONSTRAINT personaje_conjuntohabilidades_id_key UNIQUE (
        conjuntohabilidades_id);

ALTER TABLE ONLY personaje
    ADD CONSTRAINT personaje_pkey PRIMARY KEY (id);

ALTER TABLE ONLY sistema
```

```
    ADD CONSTRAINT sistema_pkey PRIMARY KEY (id);

ALTER TABLE ONLY sistema
    ADD CONSTRAINT sistema_turno_key UNIQUE (turno);

ALTER TABLE ONLY tipoalimento
    ADD CONSTRAINT tipoalimento_pkey PRIMARY KEY (tipoelemento_id);

ALTER TABLE ONLY tipoarma
    ADD CONSTRAINT tipoarma_pkey PRIMARY KEY (tipoelemento_id);

ALTER TABLE ONLY tipoelemento
    ADD CONSTRAINT tipoelemento_nombre_key UNIQUE (nombre);

ALTER TABLE ONLY tipoelemento
    ADD CONSTRAINT tipoelemento_pkey PRIMARY KEY (id);

ALTER TABLE ONLY tipoherramienta
    ADD CONSTRAINT tipoherramienta_pkey PRIMARY KEY (tipoelemento_id
        );

ALTER TABLE ONLY tipomaterial
    ADD CONSTRAINT tipomaterial_pkey PRIMARY KEY (tipoelemento_id);

ALTER TABLE ONLY tiporecurso
    ADD CONSTRAINT tiporecurso_pkey PRIMARY KEY (tipoelemento_id);

ALTER TABLE ONLY tipotrabajo
    ADD CONSTRAINT tipotrabajo_conjuntohabilidades_id_key UNIQUE (
        conjuntohabilidades_id);

ALTER TABLE ONLY tipotrabajo
    ADD CONSTRAINT tipotrabajo_nombre_key UNIQUE (nombre);

ALTER TABLE ONLY tipotrabajo
    ADD CONSTRAINT tipotrabajo_pkey PRIMARY KEY (id);

ALTER TABLE ONLY tipotrabajo_tipoherramienta
    ADD CONSTRAINT tipotrabajo_tipoherramienta_pkey PRIMARY KEY (
        tipotrabajo_id, herramientas_tipoelemento_id);

ALTER TABLE ONLY trabajo
    ADD CONSTRAINT trabajo_pkey PRIMARY KEY (id);

ALTER TABLE ONLY usomaterial
    ADD CONSTRAINT usomaterial_pkey PRIMARY KEY (id);

ALTER TABLE ONLY usomaterial
    ADD CONSTRAINT usomaterial_tipomaterial_id_tipotrabajo_id_key
        UNIQUE (tipomaterial_id, tipotrabajo_id);

ALTER TABLE ONLY usuario
    ADD CONSTRAINT usuario_email_key UNIQUE (email);

ALTER TABLE ONLY usuario
    ADD CONSTRAINT usuario_nombre_key UNIQUE (nombre);
```

```
ALTER TABLE ONLY usuario_perfil
    ADD CONSTRAINT usuario_perfil_pkey PRIMARY KEY (usuarios_id,
        perfiles_id);

ALTER TABLE ONLY usuario
    ADD CONSTRAINT usuario_pkey PRIMARY KEY (id);

ALTER TABLE ONLY localizacion_localizacion
    ADD CONSTRAINT fk1330e61fdb9bd405 FOREIGN KEY (
        localizacionespadre_id) REFERENCES localizacion(id);

ALTER TABLE ONLY localizacion_localizacion
    ADD CONSTRAINT fk1330e61ff096c015 FOREIGN KEY (
        localizacioneshijo_id) REFERENCES localizacion(id);

ALTER TABLE ONLY tipotrabajo
    ADD CONSTRAINT fk1955e7f334d071eb FOREIGN KEY (
        conjuntohabilidades_id) REFERENCES conjuntohabilidades(id);

ALTER TABLE ONLY tipotrabajo
    ADD CONSTRAINT fk1955e7f3c529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);

ALTER TABLE ONLY trabajo
    ADD CONSTRAINT fk2398d5c72240a0c9 FOREIGN KEY (localizacion_id)
        REFERENCES localizacion(id);

ALTER TABLE ONLY trabajo
    ADD CONSTRAINT fk2398d5c78bef063d FOREIGN KEY (
        personajecreador_id) REFERENCES personaje(id);

ALTER TABLE ONLY trabajo
    ADD CONSTRAINT fk2398d5c7911007cf FOREIGN KEY (
        tiporecurso_tipoelemento_id) REFERENCES tiporecurso(
        tipoelemento_id);

ALTER TABLE ONLY trabajo
    ADD CONSTRAINT fk2398d5c7b701078b FOREIGN KEY (tipotrabajo_id)
        REFERENCES tipotrabajo(id);

ALTER TABLE ONLY trabajo
    ADD CONSTRAINT fk2398d5c7f8a2a615 FOREIGN KEY (
        localizaciondestino_id) REFERENCES localizacion(id);

ALTER TABLE ONLY tipoherramienta
    ADD CONSTRAINT fk248f5582c529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);

ALTER TABLE ONLY nota
    ADD CONSTRAINT fk25240e3999d8dd FOREIGN KEY (id) REFERENCES
        elemento(id);

ALTER TABLE ONLY tipoalimento
    ADD CONSTRAINT fk2c8f0767c529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);
```

```
ALTER TABLE ONLY elemento
    ADD CONSTRAINT fk35b25132240a0c9 FOREIGN KEY (localizacion_id)
        REFERENCES localizacion(id);

ALTER TABLE ONLY elemento
    ADD CONSTRAINT fk35b25132c63236b FOREIGN KEY (personaje_id)
        REFERENCES personaje(id);

ALTER TABLE ONLY elemento
    ADD CONSTRAINT fk35b25134e6833ab FOREIGN KEY (trabajo_id)
        REFERENCES trabajo(id);

ALTER TABLE ONLY elemento
    ADD CONSTRAINT fk35b2513c529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb0072240a0c9 FOREIGN KEY (localizacion_id)
        REFERENCES localizacion(id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb0072dff56f6 FOREIGN KEY (receptor_id)
        REFERENCES personaje(id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb0074e6833ab FOREIGN KEY (trabajo_id)
        REFERENCES trabajo(id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb00761d0f3a9 FOREIGN KEY (elemento_id)
        REFERENCES elemento(id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb007911007cf FOREIGN KEY (
        tiporecurso_tipoelemento_id) REFERENCES tiporecurso(
        tipoelemento_id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb007b701078b FOREIGN KEY (tipotrabajo_id)
        REFERENCES tipotrabajo(id);

ALTER TABLE ONLY accion
    ADD CONSTRAINT fk748cb007d981789d FOREIGN KEY (actor_id)
        REFERENCES personaje(id);

ALTER TABLE ONLY tipotrabajo_tipoherramienta
    ADD CONSTRAINT fk81964d36b701078b FOREIGN KEY (tipotrabajo_id)
        REFERENCES tipotrabajo(id);

ALTER TABLE ONLY tipotrabajo_tipoherramienta
    ADD CONSTRAINT fk81964d36c61901a4 FOREIGN KEY (
        herramientas_tipoelemento_id) REFERENCES tipoherramienta(
        tipoelemento_id);

ALTER TABLE ONLY usomaterial
```

```
    ADD CONSTRAINT fk8c4715b85db7bf49 FOREIGN KEY (tipomaterial_id)
        REFERENCES tipomaterial(tipoelemento_id);

ALTER TABLE ONLY usomaterial
    ADD CONSTRAINT fk8c4715b8b701078b FOREIGN KEY (tipotrabajo_id)
        REFERENCES tipotrabajo(id);

ALTER TABLE ONLY tipoarma
    ADD CONSTRAINT fk8dab42f9c529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);

ALTER TABLE ONLY tiporecurso
    ADD CONSTRAINT fk997f8455c529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);

ALTER TABLE ONLY extraccion
    ADD CONSTRAINT fka39601b8647e28ab FOREIGN KEY (
        tipoherramienta_id) REFERENCES tipoherramienta(tipoelemento_id)
        ;

ALTER TABLE ONLY extraccion
    ADD CONSTRAINT fka39601b8a178144b FOREIGN KEY (tiporecurso_id)
        REFERENCES tiporecurso(tipoelemento_id);

ALTER TABLE ONLY personaje
    ADD CONSTRAINT fkaa4cc1c714505a4b FOREIGN KEY (usuario_id)
        REFERENCES usuario(id);

ALTER TABLE ONLY personaje
    ADD CONSTRAINT fkaa4cc1c72240a0c9 FOREIGN KEY (localizacion_id)
        REFERENCES localizacion(id);

ALTER TABLE ONLY personaje
    ADD CONSTRAINT fkaa4cc1c734d071eb FOREIGN KEY (
        conjuntohabilidades_id) REFERENCES conjuntohabilidades(id);

ALTER TABLE ONLY personaje
    ADD CONSTRAINT fkaa4cc1c779ca768c FOREIGN KEY (
        trabajoparticipa_id) REFERENCES trabajo(id);

ALTER TABLE ONLY tipomaterial
    ADD CONSTRAINT fkd792fc9bc529bc9 FOREIGN KEY (tipoelemento_id)
        REFERENCES tipoelemento(id);

ALTER TABLE ONLY usuario_perfil
    ADD CONSTRAINT fke578dc3d7bc6b05b FOREIGN KEY (perfiles_id)
        REFERENCES perfil(id);

ALTER TABLE ONLY usuario_perfil
    ADD CONSTRAINT fke578dc3db7793714 FOREIGN KEY (usuarios_id)
        REFERENCES usuario(id);

ALTER TABLE ONLY localizacion_tiporecurso
    ADD CONSTRAINT fkef9f4004d6a2884 FOREIGN KEY (
        recursos_tipoelemento_id) REFERENCES tiporecurso(
        tipoelemento_id);
```

```
ALTER TABLE ONLY localizacion_tiporecurso
    ADD CONSTRAINT fkef9f40075c27db FOREIGN KEY (localizaciones_id)
        REFERENCES localizacion(id);
```

Parte VIII

Anexo II – Primeras pruebas con el entorno

Intentaremos que el producto se pueda instalar fácilmente en cualquier entorno. Lo ideal sería empaquetar un fichero *war* que pudiera desplegarse en cualquier contenedor de *servlets* o servidor de aplicaciones. No obstante, nos encontramos con algunas complicaciones. Nos vemos obligados a configurar de alguna manera el equipo donde se desee desplegar nuestro software.

Las primeras pruebas de lo que será nuestro entorno de desarrollo van encaminadas a minimizar esa configuración y a facilitar el desarrollo lo máximo posible. De esta forma, por un lado intentaremos asegurarnos —desde el principio— de poder realizar una entrega final satisfactoria; y por el otro, dejar todo listo para que la fase de implementación sea sencilla, con el menor número de obstáculos posible, y que nos permita centrarnos en la lógica de negocio.

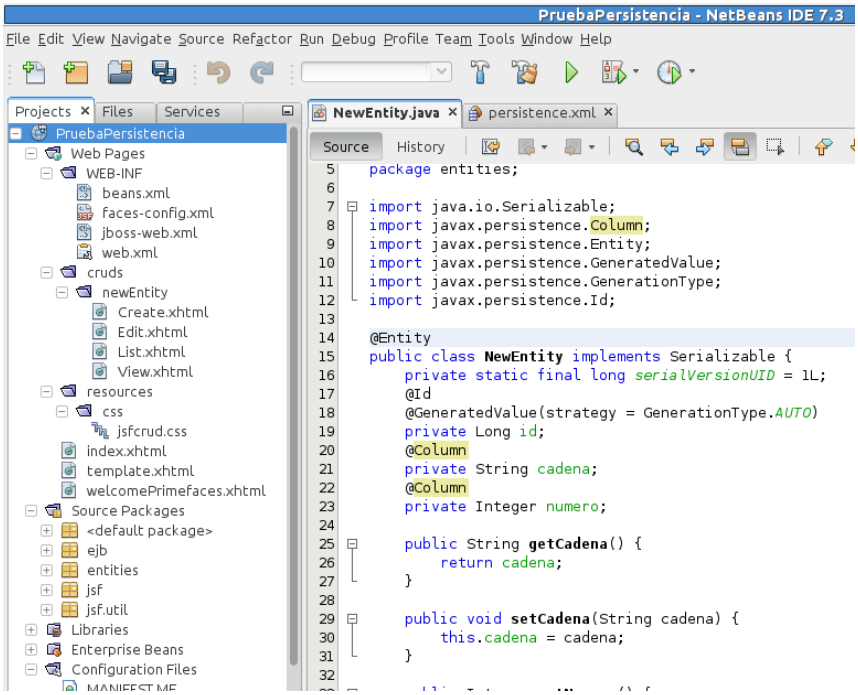


Figura 32: Para crear esta entidad, solamente hemos incluído a mano el "String cadena" y el "Integer numero", con sus anotaciones "@Column".

Herramienta de implementación

La herramienta de implementación elegida ha sido NetBeans. Si bien es posible que tenga menos capacidades que Eclipse, y que posiblemente sea menos utilizada a nivel profesional que esta última, para las tecnologías que utilizaremos incluye buenas

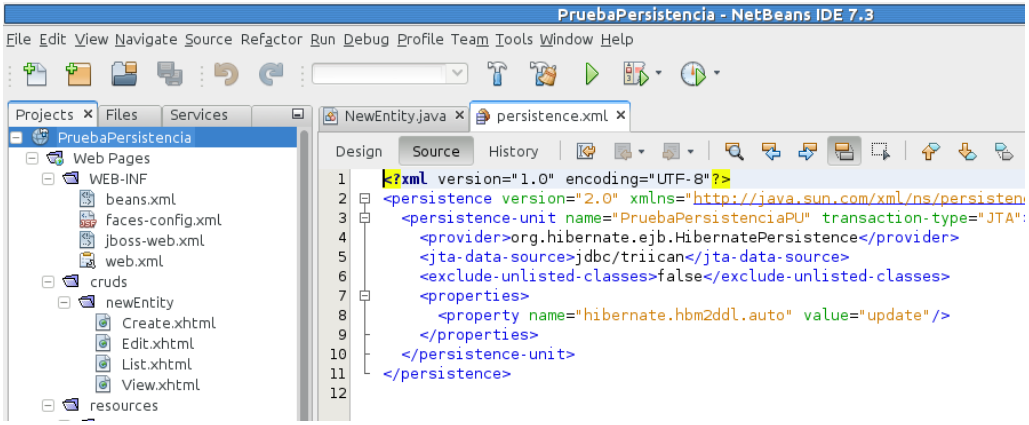


Figura 33: Fichero de configuración de la persistencia generado mediante un "mago" de Netbeans.

Figura 34: En el panel de administración del servidor de aplicaciones, configuramos la fuente de datos.

herramientas preconfiguradas y listas para funcionar de forma bastante intuitiva. Por ejemplo, "magos" de creación de los ficheros de configuración principales (véase fig. 33) o un creador de *cruds* básico a partir de las entidades, que genera desde los beans necesarios hasta las páginas (aunque muy básicas) en formato *xhtml*.

Sin entrar en discusiones, entre ambos entornos de desarrollo integrado, no obstante, parece que cualquiera de las opciones sería igualmente válida.

14. Contenedor

Al utilizar tecnologías propias de Java EE, como EJB 3 o JPA 2, nos encontramos con que cualquier contenedor de *servlets* no nos proporciona las capacidades requeridas. Es por esto que elegiremos algún servidor de aplicaciones que implemente el estándar Java EE 6, y que incluya el máximo de librerías necesarias. Puesto que el *IDE* elegido integra un servidor GlassFish 3, nos valdremos de este. Aunque también utilizaremos para algunas pruebas un servidor JBoss 7 en modo *standalone*. Con ello pretendemos asegurarnos de que el entorno de desarrollo no nos crea algún tipo de dependencia que quede fuera del fichero *war* final, o algún tipo de configuración que posteriormente no

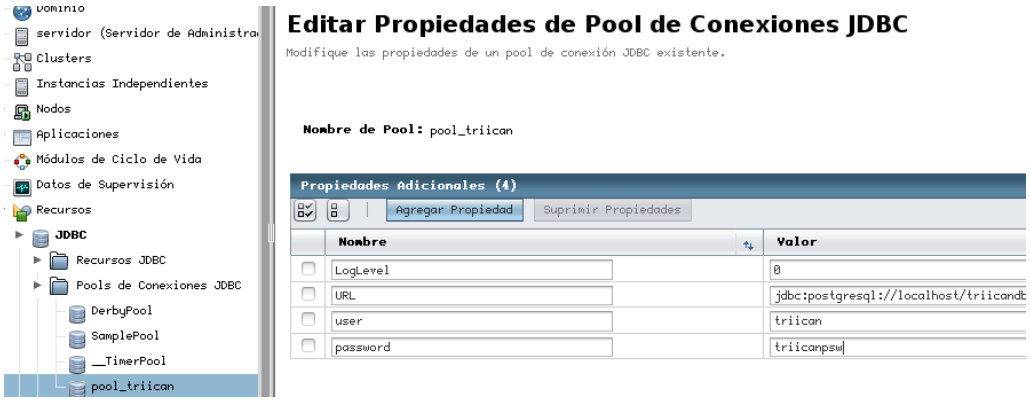


Figura 35: Especificamos los parámetros de la fuente de datos para nuestro entorno local.

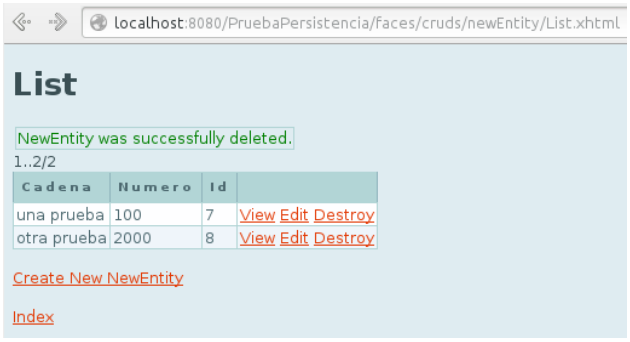


Figura 36: Listado, desde la vista en el navegador, de las entidades guardadas.

sepamos resolver manualmente.

15. Persistencia

Para la persistencia utilizaremos Hibernate. Los motivos son claros: es uno de los mapeadores objeto-relacional que implementa el estándar JPA 2 más conocidos y probados. Además de que sus librerías ya se incluyen tanto en el entorno elegido (NetBeans con GlassFish) como en el otro servidor de aplicaciones que probaremos: JBoss 7.

Con esto también intentaremos que el sistema de gestión de base de datos sea independiente de nuestro producto. Usaremos Postgresql 9 (véase fig. 38 en la página 59), pero debería ser igualmente posible arrancar el software en un sistema MySQL o cualquier otro soportado por Hibernate.

De igual manera, vamos a intentar que en el código fuente solo se especifique el nombre JNDI de la fuente de datos; y que los parámetros como URL de la conexión, nombre de usuario y contraseña sean configurados dentro del servidor de aplicaciones (véase fig. 35).

Figura 37: Creación de una nueva instancia desde la vista.

16. Anotaciones

Para automatizar al máximo la configuración, haremos uso de las anotaciones (*annotations*) tanto de JSF como de JPA (véase fig. 32 en la página 55). Esto nos permite añadir los metadatos relativos a nuestro código dentro del mismo. Así evitaremos generar o editar ficheros en formato *xml*, haciendo que el esfuerzo de desarrollo sea algo menor y más cómodo al tener toda la información de clases o interfaces fácilmente localizable.

17. Resultados

Hemos creado una entidad muy sencilla, con tan solo dos atributos (además del identificador). Añadiendo un poco de configuración para la fuente de datos en el servidor de aplicaciones (véanse fig. 34 en la página 56 y fig. 35), y referenciándola en los "magos" de NetBeans, conseguimos generar un pequeño software de prueba que cubre una funcionalidad *crud* desde la vista del usuario (fig. 36 en la página anterior y fig. 37) hasta la lectura y escritura en base de datos (fig. 38 en la página siguiente).

Como conclusión, tenemos ya la base para afrontar la fase de implementación con unas garantías mínimas.

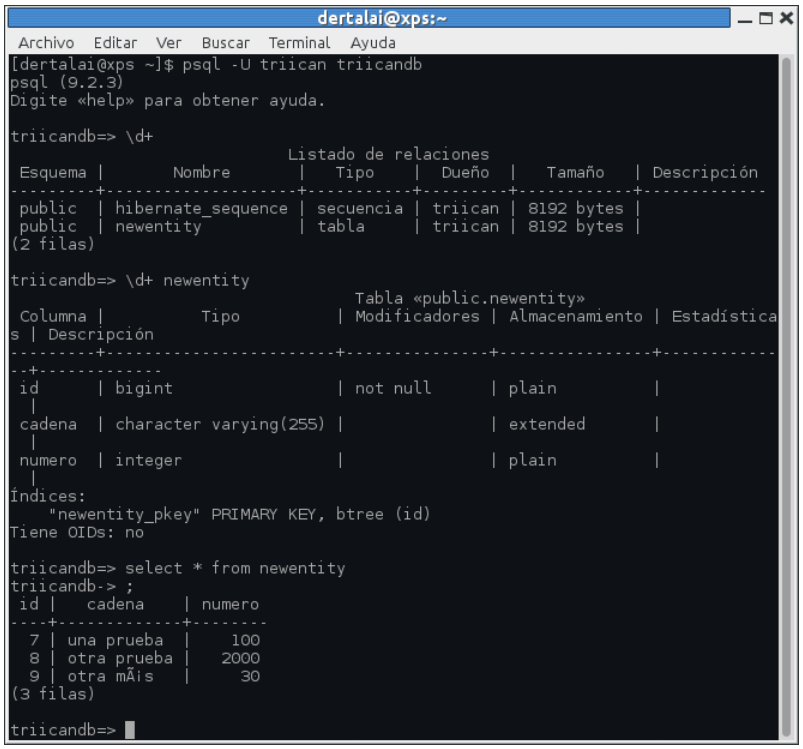


Figura 38: Comprobación de los cambios realizados, de forma automática, en base de datos.