Sistemas Informáticos

Curso 2006-2007

Aplicación de Técnicas de Inteligencia Artificial en el Sistema de Enseñanza Interactivo Javy2

Eduardo Bustos Pérez

Luis Briones Tena

Elena Pía Núñez González

Dirigido por:
Prof. Dña. Mª Belén Díaz Agudo
Dpto. Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Los abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Fdo: Eduardo Bustos Pérez

Fdo: Luis Briones Tena

Fdo: Elena Pía Núñez González

I

"Dime algo, lo olvidaré
Muéstramelo, podré recordarlo
Pero involúcrame en ello, y lo comprenderé."

Proverbio chino

"Supongamos, entonces, que la mente sea, como se dice, un papel en blanco, limpio de toda inscripción, sin ninguna idea. ¿Cómo llega a tenerlas? ¿De dónde se hace la mente con ese prodigioso cúmulo, que la activa e ilimitada imaginación del hombre ha pintado en ella, en una variedad casi infinita? ¿De dónde saca todo ese material de la razón y del conocimiento? A esto contesto con una sola palabra: de la experiencia; he allí el fundamento de todo nuestro conocimiento, y de allí es de donde en última instancia se deriva."

John Locke

Resumen

El objetivo de este proyecto ha sido la implementación de diferentes aspectos del sistema de enseñanza interactivo Javy2, que está siendo desarrollado dentro del grupo GAIA, que requerían la aplicación de técnicas de inteligencia artificial. Se han utilizado técnicas de razonamiento basado en casos para implementar un sistema que proponga ejercicios a un alumno y que además sea capaz de proporcionar ayuda contextualizada sobre el modo de resolverlos, se han utilizado sistemas de reglas para convertir el ejercicio propuesto en un escenario del juego Javy2, y por último se han utilizado técnicas de análisis sintáctico para transformar un ejercicio en Java a una representación semántica del mismo basada en una ontología.

Summary

The goal of this project has been the implementation of different aspects of the interactive learning system Javy2, that is being developed by the group GAIA, that required the application of artificial intelligence techniques. Case based reasoning techniques have been used to implement a system that proposes exercises to a student and that is also capable to offer contextualized help about the way to solve them, expert systems have been used to convert the proposed exercise in a scenary of the game Javy2, and syntactic analysis techniques have been used to translate a Java exercise into an ontology based semantic representation of that exercise.

Prólogo

La motivación principal de este proyecto es la aplicación de la Inteligencia Artificial para crear programas que imiten el comportamiento inteligente de las personas. La Inteligencia Artificial posibilita que las máquinas realicen tareas hasta ahora sólo podían ser hechas por humanos, siendo la responsable de numerosos avances en la Informática y, por extensión, en todas aquellas ciencias en las que la Informática sirve de soporte.

Otra motivación es la posibilidad de colaborar en la creación de un producto software para la enseñanza, que pretende aportar de una manera lúdica conocimientos técnicos, como es la Máquina Virtual de Java. La Informática es una ciencia con aplicaciones en numerosas áreas, como medicina o economía, pero en la enseñanza todavía no está tan extendida. Trabajar en un proyecto innovador es una motivación añadida a este proyecto.

Los trabajos y aplicaciones software de la vida real rara vez se realizan por una única persona o por un grupo pequeño de personas. Trabajar dentro de una aplicación que está siendo desarrollada por un grupo amplio de investigación, y realizando trabajo útil, es una oportunidad difícil de rechazar. Trabajar dentro de un grupo amplio supone una serie de retos que requieren la aplicación de la ingeniería del software para coordinar los esfuerzos de los diferentes miembros del grupo. Ser integrante de este proyecto es un privilegio y una oportunidad para aprender inmejorable.

Índice general

1. Introducción **1**

1.1. Descripción del proyecto 1

1.2. Requisitos iniciales 3

1.3. Requisitos finales 7

2. Ayuda Sintáctica **9**

2.1. Objetivo 9

2.2. Introducción 10

2.3. Adquisición de conocimiento general 11

2.3.1. Razonamiento basado en casos 12

2.3.2. Razonamiento basado en casos textuales 15

2.3.3. Similitud del coseno 17

2.3.4. Mejoras de sistemas CBR Textuales 18

2.4. Estudio sobre las herramientas empleadas 19

2.4.1. Proyecto de SSII de 2005-2006 19

2.4.2. jColibri 22

2.4.3. ScriptSystem 28

2.4.4. Javy2 31

2.4.5. Tortoise SVN 35
2.5. Desarrollo 35
2.5.1. Estructura General 35
2.5.2. La clase Help 37
2.5.3. Módulos de ayuda 38
2.5.4. Conexión con jColibri 40
2.5.5. Clases de apoyo 41
2.5.6. Conexión con Javy2 42
2.6. Creación de textos de ayuda 44
2.7. Recomendaciones en la creación de textos 46
2.8. Mantenimiento 47
2.9. Trabajo futuro 48
2.10. Conclusiones 49

3. Ayuda Semántica **51**
3.1. Objetivo y motivación 51
3.2. Introducción 52
3.3. Adquisición de conocimiento sobre el juego y la metáfora 53
3.3.1. Conocimiento adquirido sobre la metáfora . . . 54
3.3.2. Conocimiento adquirido sobre los controles . . . 56
3.4. Adquisición de conocimiento sobre las herramientas . . 60
3.4.1. Conocimiento adquirido sobre ontologías 60
3.4.2. Ontobridge 67
3.4.3. Wordnet 69
3.5. Desarrollo 69

3.5.1. Desarrollo de la ontología 70
3.5.2. Ayuda semántica en Java 81
3.5.3. Función de similitud 82
3.6. Ejemplos de ejecución 86
3.7. Trabajo Futuro 87
3.8. Conclusiones 90

4. Evaluación de las Ayudas Semántica y Sintáctica 92
4.1. Evaluación realizada por nosotros 93
4.2. Evaluación por gente externa a la implementación . . . 96
4.3. Conclusiones obtenidas de la Evaluación 99

5. Recomendador de ejercicios 102
5.1. Objetivo y motivación 102
5.2. Estructura general 103
5.3. Obtención de Base de Ejercicios Java 105
5.4. Módulo Herramienta Java2Owl 107
5.4.1. Objetivo y motivación 107
5.4.2. Adquisición general de conocimiento 109
5.4.3. Estructura de clases 110
5.4.4. Funcionamiento y ejemplos (Validación) 116
5.4.5. Trabajo Futuro 124
5.5. Editor de perfil de usuario 126
5.6. Manejador de perfiles de usuario: 129
5.6.1. Descripción del perfil de usuario 129
5.6.2. Organización 129

5.7. Recomendador CBR 130

5.7.1. Inserción del perfil de usuario en OWL y Funciones de similitud 132

5.7.2. Desarrollo 139

5.8. Conclusiones 147

6. Generador de Escenarios 149

6.1. Objetivo 149

6.2. Estructura General 151

6.2.1. Parámetros de entrada 152

6.2.2. Análisis del ejercicio. Escenario básico 153

6.2.3. Obtención del perfil del alumno 153

6.2.4. Reglas de generación. Escenario Variado 153

6.2.5. Guardado del escenario generado 156

6.3. Validación 157

6.4. Mantenimiento 157

6.5. Ampliaciones 157

6.6. Conclusiones 158

7. Conclusiones 159

A. Apéndice de la Ayuda Sintáctica 161

A.1. Signatura de los métodos Java invocados desde C++ . 161

A.2. Sistema de archivos de la ayuda sintáctica 162

B. Apéndice de la Ayuda Semántica 165

B.1. Estructura general 165

B.1.1. El paquete motor 165

C. Apéndice de la Evaluación de las Ayudas **171**

D. Apéndice de la Herramienta Java2Owl **179**

Bibliografía **182**

Glosario de Términos **185**

Capítulo 1

Introducción

1.1. Descripción del proyecto

La mejor manera de enseñar una materia ha sido fuente de discusión en la didáctica durante innumerables años, pero es ampliamente aceptado que existen ciertos conocimientos que son mejor aprendidos mediante su práctica. Este concepto, llamado *"learning by doing"* no es nuevo, y ya se puede vislumbrar por ejemplo en el proceso de aprendizaje de profesiones tradicionales como la de carpintero o herrero[B12], y más recientemente en los simuladores de vuelo. Con la llegada de las nuevas tecnologías y el auge de los videojuegos se ha introducido un nuevo concepto derivado de éste, el *"learning by gaming"* que aprovecha la capacidad lúdica de los videojuegos para entretener al jugador a la par que se le enseñan nuevos conocimientos[B13]. Es en este campo donde se sitúa Javy2, como un juego que enseña conceptos de compilación de Java.

Este proyecto forma parte del sistema de enseñanza interactivo Javy2 desarrollado por el grupo de investigación GAIA. Javy2 es un juego mediante el cual se enseña la compilación de Java. Este proyecto está formado por 5 módulos de diferente funcionalidad que se integran dentro de Javy2.

El módulo de ayuda sintáctica proporciona a Javy2 la capacidad de ofrecer ayuda al usuario. Puesto que el usuario es a la vez jugador y alumno, la ayuda proporcionada versa tanto sobre el juego como sobre la materia que se le desea enseñar. Este módulo está implementado

usando un sistema de razonamiento basado en casos textuales. Mediante el uso de técnicas de recuperación de información, en concreto técnicas de análisis sintáctico, se proporciona al usuario una respuesta a una pregunta efectuada en lenguaje natural.

El módulo de ayuda semántica también proporciona a Javy2 la capacidad de ofrecer ayuda sobre el juego al usuario. Este módulo está implementado usando una ontología del conocimiento sobre Javy2. Mediante el análisis semántico de la pregunta efectuada por el usuario se le proporciona a éste la respuesta más adecuada a dicha pregunta.

Mediante el análisis comparativo de los dos módulos de ayuda se ha decidido cuál de los módulos era el más apropiado para Javy2. El análisis de los resultados de la evaluación de los módulos nos ha llevado a concluir que la mejor opción era implementar la ayuda sobre la compilación de Java mediante análisis sintáctico y la ayuda sobre el juego mediante análisis semántico.

El módulo de la herramienta Java2Owl proporciona la capacidad de traducir un ejercicio en el lenguaje de programación Java a una representación semántica de ese ejercicio en OWL. Al traducir los ejercicios a una representación semántica se posibilita el uso de técnicas de análisis semántico para recomendar los ejercicios apropiados a un alumno.

El módulo del recomendador de ejercicios proporciona a Javy2 la capacidad de proponer al jugador el ejercicio más adecuado a los conocimientos de ese jugador. Mediante el análisis semántico del perfil del alumno y de los ejercicios de la base de casos, previamente traducidos por la herramienta Java2Owl, se escoge el ejercicio más adecuado a los conocimientos del alumno.

Javy2 se basa en una representación metafórica de escenarios que representan ejercicios. En cada escenario el alumno actúa aprendiendo a compilar el ejercicio en la máquina virtual de Java (JVM).

Una vez obtenido el ejercicio que se le va a proponer al usuario hay que generar el escenario que representa dicho ejercicio en Javy2, ésto es lo que hace el módulo del generador de escenarios. Tomando el ejercicio propuesto, ya compilado, genera el escenario que representa ese ejercicio

y posteriormente lo varía siguiendo una serie de reglas que toman en cuenta al jugador actual para hacerlo más o menos difícil.

Figura 1.1: Imagen del juego Javy2

1.2. Requisitos iniciales

Inicialmente no sabíamos con exactitud qué es lo que Javy2 necesitaba, únicamente teníamos claro que necesitaba un módulo de ayuda, aunque teníamos algunas ideas sobre lo que se podía realizar.

Inicialmente establecimos que serían requisitos de nuestro proyecto la creación de 4 módulos.

1. **Módulo 1 - Ayuda:** Este módulo se podía dividir a su vez en dos; el primero sería la ayuda que afecta al dominio de la JVM, y el segundo el que afecta al dominio del juego en sí mismo. Para los dos tipos de ayuda nos propusimos en un principio que funcionasen de manera reactiva, y si hubiera tiempo extender su funcionamiento a proactiva.

   - **Ayuda sobre JVM:** Nuestro propósito será el de adaptar y conectar la parte concerniente al proyecto del curso 2005/06 *"Uso de CBR en sistemas de enseñanza interactiva"*[B1]. Por

lo que nuestro principal requisito será el de hacer funcionar este sistema dentro de la implementación de Javy2.

- **Ayuda sobre el estado del juego:** Esta parte del módulo tendrá los siguientes requisitos:
  - Identificar la experiencia del jugador para que la calidad de la ayuda proporcionada al usuario sea adecuada a dicha experiencia.
  - La ayuda funcionará comprendiendo las peticiones del usuario en lenguaje natural.
  - Se podrán proporcionar pistas para solucionar el ejercicio planteado al jugador.
  - La ayuda tendrá una sección para describir los comandos del juego.
  - También será capaz de explicar el desarrollo del juego, así como describir la utilidad de los diferentes escenarios.
  - Será una ayuda ampliable a otros idiomas diferentes al inglés.
  - Independiente de la GUI.

2. **Módulo 2 - Malos:** Con Malos nos referíamos a implementar la inteligencia artificial de los portadores de los diferentes recursos del juego. Es decir los personajes que se moverán por los diferentes escenarios intentando escaparse del usuario. Una pequeña aproximación de los requisitos que tendrá este módulo sería:

   - Capacidad para moverse tanto de manera independiente como formando parte de una manada.
   - Evitar paredes y posibles obstáculos que existan en el escenario.
   - Esconderse y escaparse del jugador.

3. **Módulo 3 - Elección de ejercicios:** En este módulo pretendemos realizar un conjunto de ejercicios para poder proponer al jugador como reto.

   - Este sistema tendrá que identificar previamente la experiencia del jugador para así proponer unos ejercicios concordes con el conocimiento adquirido.

4. **Módulo 4 - Jugador inteligente:** En este módulo, siempre y cuando tuviésemos tiempo suficiente, pretendemos implementar o realizar una aproximación a la implementación de lo que será el

jugador inteligente del juego. Es decir el personaje con el que un usuario pueda competir a lo largo de una partida.

- Tendrá diferentes niveles de habilidad
- Tendrá que simular lo mejor posible el funcionamiento de un jugador real de Javy2.

Pronto se vio que muchos de estos requisitos no eran viables, por ejemplo la creación del jugador inteligente estaba supeditada a que Javy2 estuviese prácticamente completado, lo que no iba a ser posible. El módulo que implementaba el comportamiento de los portadores de recursos puesto que no tenía una aplicación con un fuerte requisito de Inteligencia Artificial y además podría formar parte del trabajo de otra persona del grupo de desarrollo.

Poco a poco se fueron concretando los requisitos y además de eliminar los módulos que no se iban a poder realizar se añadieron nuevos módulos como el editor de escenarios, que finalmente se concretaría en el generador de escenarios.

La segunda versión de los requisitos constaba de 3 módulos y era más concreta, especialmente en la definición de los requisitos de la ayuda.

1. **Módulo 1 - Ayuda:** Este módulo se puede dividir a su vez en dos; el primero sería la ayuda que afecta al dominio de la JVM, y el segundo el que afecta al dominio del juego en sí mismo. Para los dos tipos de ayuda nos hemos propuesto que funcionen de manera reactiva.
   - **Ayuda sobre JVM:** Nuestro propósito será el de adaptar y conectar la parte concerniente al proyecto del curso 2005/06 *"Uso de CBR en sistemas de enseñanza interactiva"*[B1]. Por lo que nuestro principal requisito será el de hacer funcionar este sistema dentro de la implementación de Javy2.
   - **Ayuda sobre el estado del juego:** Esta parte del módulo tendrá los siguientes requisitos:
     - La ayuda funcionará comprendiendo las peticiones del usuario en lenguaje natural.
     - Se podrán proporcionar pistas para solucionar el ejercicio planteado al jugador.

- La ayuda tendrá una sección para describir los comandos del juego.
- También será capaz de explicar el desarrollo del juego, así como describir la utilidad de los diferentes escenarios.
- Será una ayuda ampliable a otros idiomas diferentes al inglés.
- Independiente de la GUI.
- Implementaremos dos tipos de ayuda sobre el juego:
  - **Ayuda sintáctica:** Es así como está hecho el módulo de ayuda sobre la JVM. En un preciclo (antes de la primera cuestión) todos los textos son tratados por jColibri eliminando artículos, preposiciones, etcétera; devolviendo un vector de palabras. Cada pregunta del usuario se trata de igual manera, y se pasa a buscar las máximas similitudes entre palabras del texto de la pregunta con los textos de ayuda, devolviendo un vector ordenado con las respuestas.
  - **Ayuda semántica:** Para ello creamos una ontología sobre Javy2. Cada concepto de nuestra ontología está relacionado con un texto de ayuda por medio de una URL, que contiene la ruta del fichero con dicho texto de ayuda. Este método proporciona una mayor independencia entre textos-conceptos, que facilita posibles modificaciones o la reutilización para, por ejemplo, cambiar el idioma de la ayuda.

    Para proporcionar una ayuda más eficaz asociamos a cada concepto nuevas etiquetas con sinónimos del nombre de éste y distintas palabras que se pueden asociar al texto. Así, por ejemplo, a la ayuda del módulo "malos" se puede llegar por medio de "malo", "enemigo", "porta-recursos"... A la hora de relacionar los textos con la ontología contamos con herramientas semánticas, entre las que tenemos SMORE y AKTIVE MEDIA. Buscamos alguna herramienta más, y de entre todas estudiamos ventajas e inconvenientes, escogiendo la que más nos interese.

2. **Módulo 2 - Recomendador de Ejercicios:** En este módulo pretendemos relacionar un conjunto de ejercicios para poder proponer al jugador como reto.
   - Este sistema tendrá que identificar previamente la experien-

cia del jugador para así proponer unos ejercicios concordes con el conocimiento adquirido.

- Será un sistema CBR que se basará en una base de casos en xml q conste de distintos ejercicios junto con su dificultad en cada aspecto.
- Relacionando la experiencia del jugador con las dificultades de cada ejercicio se intentará recomendar el más adecuado.
- Nos intentaremos servir de algunas partes del proyecto del año pasado.

3. **Módulo 3 - Editor de Escenarios:** En Javy2 hay un editor de mapas que a través de un formato específico traduce los distintos elementos del mapa en clases C++. Cada escenario corresponde con un ejercicio.

   Por cada ejercicio hay un fichero xml q contiene el código Java y la idea es añadir cómo se representa en el escenario dicho ejercicio (los recursos necesarios, los tipos de malos que portarán cada recurso...).

   Con esto se quiere permitir:

   a) Según el perfil de usuario seleccionar adecuadamente el ejercicio. Para ello necesitamos una ontología sobre conceptos de compilación que ya está creada.

   b) Practicar. Si un usuario quiere practicar sobre un tipo de ejercicio, poder generar automáticamente variaciones sobre dicho ejercicio.

   c) Creación automática de lo ficheros xml a partir del escenario creado.

1.3. Requisitos finales

La mayoría de los requisitos iniciales se cumplieron, aunque hubo algunos que no se pudieron realizar, mientras que aparecieron otros nuevos que no estaban inicialmente. El caso más claro de nuevos requisitos es la aparición del módulo de la herramienta Java2Owl, que inicialmente no estaba prevista. Esta herramienta se creó por necesidad. Para poder recomendar un ejercicio era necesario poder analizarlo, y la mejor forma era semánticamente usando la ontología sobre Java,

pero ningún ejercicio estaba en esta representación, por lo que fue necesaria la creación de una herramienta para transformar los ejercicios a esta representación semántica.

A continuación analizamos los requisitos iniciales que no se cumplieron o que cambiaron.

- **Módulo 1 - Ayuda:**
  - No se proporcionan pistas para la resolución del ejercicio puesto que esto supondría llevar constantemente actualizado el estado del juego, y debido a que todavía sufre demasiados cambios, este trabajo sería inútil.
  - No se utilizó ninguna de las herramientas semánticas como SMORE o AKTIVE MEDIA pues no fueron necesarias.
- **Módulo 2 - Herramienta Java2Owl:** Este módulo no estaba previsto en los requisitos iniciales. Esto módulo tiene los siguientes requisitos:
  - Transformará un ejercicio en Java en la representación semántica de OWL.
  - Empleará el traductor de Java a XML de Harsh Jain para transformar primero a XML y de XML a OWL.
  - Podrá guardar la representación en OWL de diferentes ejercicios en el mismo fichero.
- **Módulo 3 - Recomendador de ejercicios:**
  - La base de casos no estará en xml, será una representación en OWL de los ejercicios.
- **Módulo 4 - Generador de escenarios:**
  - La ontología sobre conceptos de compilación no fue necesaria en la generación de escenarios, fue necesaria en la recomendación de ejercicios. La ontología creada no era válida por lo que fue necesario crear una nueva, aunque no por nosotros, esta siendo desarrollada por otro miembro del grupo de desarrollo.

Capítulo 2

Ayuda Sintáctica

2.1. Objetivo

En cualquier juego es necesario un sistema de ayuda para que el usuario comprenda rápidamente los conceptos básicos del juego y para que no se quede bloqueado. De esta forma se aumenta la jugabilidad del juego, haciéndolo más interesante para el jugador, y el objetivo máximo de un juego es la satisfacción del jugador.

Javy2 es un juego pero también es un sistema interactivo de enseñanza. Los juegos necesitan un sistema de ayuda para aumentar la posibilidad de que al jugador le guste el juego, pero los sistemas interactivos de enseñanza deben ser capaces además de proporcionar al usuario las explicaciones necesarias para que éste entienda y aprenda los conceptos que el sistema le está intentando enseñar. Ambos objetivos se complementan, pues al aprender los conocimientos que el sistema enseña, el jugador disfrutará más con el juego, y al disfrutar con el juego, el usuario o jugador aprenderá más rápidamente los conceptos que enseña el juego.

Nuestro sistema estará enfocado a proporcionar ayuda al usuario, el jugador de Javy2, tanto sobre los conceptos a aprender (compilación de Java e instrucciones de la JVM) como sobre el desarrollo del juego (mecanismos del juego, metáfora del juego, etc.).

El objetivo de este módulo consiste en proporcionar ayuda al usuario

sobre la máquina virtual de Java o sobre el juego Javy2. El usuario introducirá una pregunta en un terminal del juego Javy2 en lenguaje natural y el sistema deberá proporcionarle la ayuda adecuada a su pregunta. Puesto que el juego está en desarrollo el sistema debe ser altamente adaptable, por ejemplo, para añadir nuevos idiomas de ayuda, y también debe facilitar los cambios, de manera que si se cambia algún aspecto del juego los cambios que se deban hacer en el módulo sean mínimos, esto lo haremos separando el conocimiento de los mecanismos de búsqueda de información.

Este módulo se integrará en el proyecto de Javy2, que esta programado en C++, mientras que vamos a realizar este módulo en el lenguaje de programación Java, trataremos más adelante la manera de unir ambas partes mediante el interfaz JNI.

La parte de este módulo que proporciona ayuda sobre el juego Javy2 se comparará con la implementada en el módulo de ayuda semántica, quedando finalmente la que mejores resultados ofrezca tras la evaluación de ambas.

2.2. Introducción

Como ya hemos comentado, Javy2 es un sistema de enseñanza interactivo de la JVM. Dentro de Javy2 este módulo de ayuda se encarga de proporcionar una guía al aprendiz para resolver sus dudas, así como una ayuda al jugador para que pueda completar satisfactoriamente el juego.

La primera parte de este módulo, la que proporciona ayuda sobre la JVM, formaba parte de un proyecto previo de Sistemas Informáticos[B1], por lo que extrajimos gran parte de lo necesario de dicho proyecto, aunque después hubo que adaptarlo a nuestro proyecto.

La segunda parte de este módulo, la que proporciona ayuda sobre el juego Javy2, se realizó desde el principio, aunque basándonos en la primera parte del módulo.

Ambas partes se realizaron siguiendo el esquema de similitud sintáctica de textos, aunque no hubimos de implementarlo pues para ello em-

pleamos el sistema jColibri[B2] desarrollado por el grupo GAIA de la UCM. jColibri proporciona un framework para la creación de aplicaciones CBR, teniendo además una extensión específica para la creación de aplicaciones CBR textuales [B11], lo que nos resultó tremendamente útil.

El sistema, como CBR textual[1], obtendrá una pregunta formulada por el jugador de Javy2, la comparará con una base de casos, obteniendo la respuesta a la pregunta inicialmente formulada, y finalmente mostrándosela al usuario.

La primera tarea que tuvimos que realizar fue la de profundizar nuestro conocimiento tanto en el esquema de similitud de textos que íbamos a seguir como en las herramientas que íbamos a emplear para implementarlo.

Una vez adquirido dicho conocimiento extrajimos las partes del proyecto de Sistemas Informáticos de 2005-2006 que nos resultarían útiles, observamos como se servía de jColibri para proporcionar la ayuda, y finalmente lo adaptamos a nuestro proyecto.

Una vez realizada la primera parte del módulo implementamos la segunda siguiendo el mismo diseño que el realizado en la primera. En esta parte tuvimos que adquirir el conocimiento para realizar además los textos que conformarían la ayuda sobre el juego Javy2.

En este capítulo se explicará la adquisición de conocimiento que hubo que realizar para entender el problema a resolver y para entender las herramientas que se iban a emplear en la resolución de dicho problema. Se explicará también el desarrollo del módulo, el trabajo futuro que se puede realizar y las conclusiones que se extrajeron del trabajo.

2.3. Adquisición de conocimiento general

La información adquirida que mostramos en este apartado es un resumen de parte de lo aprendido en las asignaturas de "Inteligencia Artificial e Ingeniería del Conocimiento" (IAIC) y "Ingeniería de Sistemas

[1]El concepto de CBR textual se explica más adelante en la sección 2.3.2.

Basados en Conocimiento" (ISBC). También se han consultado otras fuentes como el proyecto de Sistemas Informáticos del año pasado[B1] que trataba sobre este tema (como veremos más adelante), varios libros como [B4] o [B7], y diversas páginas web con información relevante.

2.3.1. Razonamiento basado en casos

¿Qué es el razonamiento basado en casos? Básicamente, el razonamiento mediante el cual resolvemos un nuevo problema recordando una situación previa similar a la actual y reutilizando o adaptando dicha información para resolver el nuevo problema.

"Case-Based reasoning o CBR (Razonamiento basado en casos) es el proceso de solucionar nuevos problemas basándose en las soluciones de problemas anteriores. Un mecánico de automóviles que repara un motor por que recordó que otro auto presentaba los mismos síntomas está usando case-based reasoning. Un abogado que apela a precedentes legales para defender alguna causa está usando case-based reasoning. También un ingeniero cuando copia elementos de la naturaleza, está tratando a esta como una "base de datos de soluciones". El Case-Based reasoning es una manera razonar haciendo analogías. Se ha argumentado que el case-based reasoning es más que un método poderoso para el razonamiento de computadoras (computer reasoning) sino que es usado por las personas para solucionar problemas cotidianos. Más radicalmente se ha sostenido que todo razonamiento es Case-Based por que está basado en la experiencia previa."[B5]

Las principales tareas que todos los métodos CBR tienen que resolver son: identificar la situación actual del problema, encontrar un caso previo similar al nuevo, usar ese caso para proponer una solución al problema actual, evaluar la solución propuesta, y actualizar el sistema aprendiendo de esta experiencia. En el mayor nivel de abstracción, el ciclo CBR genérico puede ser descrito por los siguientes 4 procesos [B7]:

1. **Recuperación:** Recuperar el caso o casos más similares.
2. **Reutilización:** Reutilizar la información de ese caso/s para resolver el problema.
3. **Revisión:** Revisar la solución propuesta.

4. **Retención:** Retener las partes de esta experiencia que probablemente sean útiles para resolver un futuro problema.

Un nuevo problema es resuelto recuperando uno o más casos previos, reutilizar dicho caso de alguna forma, revisar la solución obtenida al reusar el caso previo, y retener o aprender la nueva experiencia incorporándola a la base de conocimiento (base de casos) existente. Cada uno de los 4 procesos implica una serie de pasos más específicos, que describiremos más adelante. En la figura 2.1, se ilustra el ciclo CBR[B7].

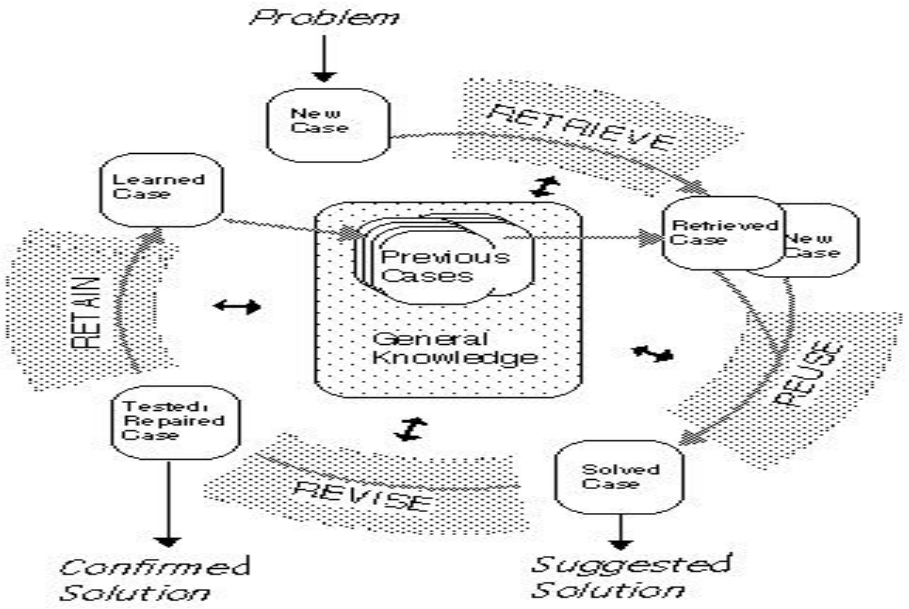

Figura 2.1: Ciclo CBR [B7]

Una descripción inicial del problema define un nuevo caso. Este nuevo caso es usado para recuperar un caso de la colección de casos previos. El caso recuperado es combinado con el nuevo caso (reutilización) para formar un caso resuelto, por ejemplo una solución propuesta al problema inicial. En el proceso de revisión esta solución es comprobada, ya sea aplicándola al mundo real o mediante la evaluación de un "profesor", y reparada si fracasó. En el proceso de retención o aprendizaje, la experiencia útil se almacena para una futura reutilización, y la base de casos es actualizada con el nuevo caso aprendido o con una modificación de casos ya existentes[B6].

El proceso de recuperación de los casos similares es el más impor-

tante, ya que este establecerá la medida de la corrección de nuestro sistema. De poco servirá un proceso de adaptación o reutilización si el caso recuperado es completamente diferente de nuestro problema. Del mismo modo, un buen proceso de aprendizaje no servirá de mucho si posteriormente no podemos recuperar correctamente los casos aprendidos similares al nuevo problema. Para tener un buen proceso de recuperación es importante tener una buena definición de los casos y elegir correctamente la función de similitud entre los casos.

Existen múltiples formas de definir un caso, tantas como problemas podamos definir, y, aunque ha habido intentos de estandarizar la representación de los casos[2], actualmente la estructura de los casos depende por completo del problema que queremos especificar. Un problema se puede especificar, entre otras formas, como una serie de características que definen un problema, obteniendo un caso estructurado. Otra forma de especificar el problema sería por ejemplo mediante un texto con la descripción del problema, de esta forma nuestro caso sería un texto.

Por ejemplo podríamos tener un CBR que nos recomendase pisos para alquilar. Una posible representación de los casos sería mediante una serie de características como metros cuadrados, precio, número de dormitorios, etc. Esta representación daría lugar a una serie de casos estructurados en los que un caso podría ser (100 m^2, 750€, 3,...). Mientras que si nuestros casos estuviesen representados por textos un posible caso podría ser "Es un piso de 100 metros cuadrados que cuesta 750 euros al mes, tiene tres dormitorios...". Según el tipo de problema al que nos enfrentásemos sería mejor escoger una representación u otra.

Los sistemas CBR tienen su origen en el trabajo de Roger Schank en la Universidad de Yale a principios de los 1980's, y ya entonces se crearon la primeras aplicaciones como CYRUS[B16] e IPP[B17]. La tecnología CBR ha ido desarrollándose y se ha aplicado con éxito en diversas áreas como el razonamiento legal o los recomendadores de productos. Como ejemplos exitosos podemos destacar el sistema Lockheed's CLAVIER[B18] utilizado para presentar a las piezas compuestas que se cocerán en un horno industrial de convección, o la aplicación help-desk como Compaq SMART system[B19].

Para una introducción completa sobre técnicas y aplicaciones del CBR

[2]Se puede ver una representación XML de los casos en [B15]

el lector interesado puede consultar [B7], [B26], [B27] o [B28].

2.3.2. Razonamiento basado en casos textuales

El razonamiento basado en casos textuales o CBR textual es un tipo específico de CBR en el que los casos son textos. Al ser los casos textos tenemos dos opciones para tratar con ellos, convertirlos a casos estructurados convirtiendo nuestro CBR textual en un "CBR tradicional", o utilizar diferentes técnicas de recuperación de información (information retrieval (IR)) para tratar directamente los textos. La conversión a un caso estructurado suele dar buen resultado siempre que podamos realizarla. En el ejemplo anterior de la recomendación de pisos no debería ser muy complicado analizar el texto "Es un piso de 100 metros cuadrados que cuesta 750 euros al mes, tiene tres dormitorios..." y extraer las características importantes como los m^2, el precio o el número de dormitorios. Sin embargo, si tratásemos por ejemplo con un resumen de un libro sería muy difícil obtener unas características que lo definiesen, por lo tanto en ocasiones es necesario tratar directamente con los textos.

La semejanza de un caso con otro viene determinada por la función de similitud entre ambos casos. Con los casos estructurados lo que se suele hacer es establecer similitud entre sus diferentes características, por lo general de valores numéricos o booleanos, y luego hacer una media o media ponderada de las similitudes de todas sus características, aunque existen bastantes más funciones de similitud que pueden ser aplicables. El caso de la similitud entre dos textos es más complicado, y se suele emplear el modelo de espacio vectorial, este modelo, en concreto la similitud del coseno en el modelo de espacio vectorial lo veremos más en detalle en la sección 2.3.3. Este modelo, aunque no es la única función de similitud aplicable, es la más empleada.

Podemos distinguir dos usos mayoritarios de estos sistemas, el de clasificación o el de respuesta.

En los CBR textuales usados como clasificación el usuario introduce una consulta, típicamente un documento, y el sistema analizando la base de casos devuelve la clasificación de dicho documento. Un ejemplo típico sería el de un filtro de spam. La base de casos estaría formada por una serie de correos clasificados como spam o ham (no spam), y la

consulta sería un nuevo correo que habríamos de clasificar como spam o ham, dependiendo de los n correos más similares de la base de casos. Por supuesto esta es una primera aproximación simple a la realización de un filtro spam mediante técnicas CBR[3]

En los CBR textuales usados para responder preguntas la mecánica es diferente, no tenemos los documentos de la base de casos clasificados según tipo, pues ese no es el objetivo. Dada una consulta en lenguaje natural, típicamente una pregunta o serie de palabras clave, el objetivo es devolver la respuesta a dicha pregunta, que puede ser una respuesta directa o una colección de documentos con información relevante a dicha pregunta. Esta es la forma básica en la que trabajan por ejemplo los motores de búsqueda en la World Wide Web[4].

Es interesante distinguir estos dos usos pues a la hora de proporcionar una respuesta no es lo mismo si pretendemos proporcionar una respuesta directa, en la que solo podemos escoger uno de nuestros casos, que si pretendemos clasificar un documento, en el que podemos por ejemplo seleccionar los n casos más similares y realizar una votación entre ellos para ver como se clasifica finalmente el documento.

Identificando los 4 procesos generales de un sistema CBR en nuestro sistema CBR textual orientado a respuesta podemos establecer las siguientes tareas a realizar:

- **Recuperación:** Recuperar el texto más similar a la consulta.
- **Reutilización:** En nuestro caso no existe adaptación, pues mostramos directamente el texto recuperado como respuesta.
- **Revisión:** La revisión corre a cargo del usuario, que establecerá si la respuesta ofrecida es válida o no según su nivel de satisfacción. No existe manera de reparar el caso estrictamente hablando, aunque el mecanismo de *more like this* permite al usuario solicitar la siguiente respuesta más similar si la mostrada no ha sido satisfactoria.
- **Retención:** En nuestro sistema no existe aprendizaje propiamente dicho, aunque la experiencia del equipo de desarrollo desem-

[3]Se pueden ver otras aproximaciones en el trabajo de Sarah Jane Delany (Dublin Institute of Technology) y Derek Bridge (University College Cork).

[4]Para más detalles ver [B4] Capítulo 23.2

peña un factor crucial al seleccionar la base de casos con la que se trabajará.

2.3.3. Similitud del coseno

Uno de los mayores problemas en la aproximación explicada anteriormente es como establecer que un texto es similar a otro. Cada uno de los métodos empleados para resolver este problema implementa una función de similitud diferente. Existen muy diversas aproximaciones para realizar esta función de similitud, como por ejemplo el uso de "features" para caracterizar un texto o la similitud basada en compresión[5], pero nosotros analizaremos a continuación la similitud del coseno.

La similitud del coseno se basa en las propiedades de vectores en un espacio euclídeo. Mide el coseno del ángulo de dos vectores en un espacio N-dimensional, y su expresión es la siguiente[B14]:

$$sim(q, D) = \frac{\sum(p_{iq} * p_{iD})}{\sqrt{\sum p_{iq}^2 * \sum p_{iD}^2}}$$
$$p_{iq} = N_{iq} * \log \frac{D_T}{D_i}$$
$$p_{iD} = N_{iD} * \log \frac{D_T}{D_i}$$

Siendo:

$sim(q, D)$ = Similitud entre la consulta q y el documento D.
p_{iq} = El peso del término i en la consulta q.
p_{iD} = El peso del término D en el documento D.
N_{iq} = El número de veces que aparece el término i en la consulta q.
N_{iD} = El número de veces que aparece el término i en el documento D.
D_T = El número de documentos de la base de casos.
D_i = El número de documentos en los que aparece el término i.

A primera vista parece complicado realizar una implementación eficiente de esta similitud puesto que hay que contar cuántas veces aparece un término en un documento o en cuántos documentos aparece un término,

[5] Ver el trabajo de Sarah Jane Delany (Dublin Institute of Technology) y Derek Bridge (University College Cork)

pero la implementación con lista invertida de términos es muy conocida y eficiente. Esta implementación es la mostrada en la figura 2.2, formada por dos tablas hash, una para los documentos, en la que se almacenan los términos y el número de apariciones en ese documento, y otra para los términos, en la que se almacenan los documentos en los que aparece cada término.

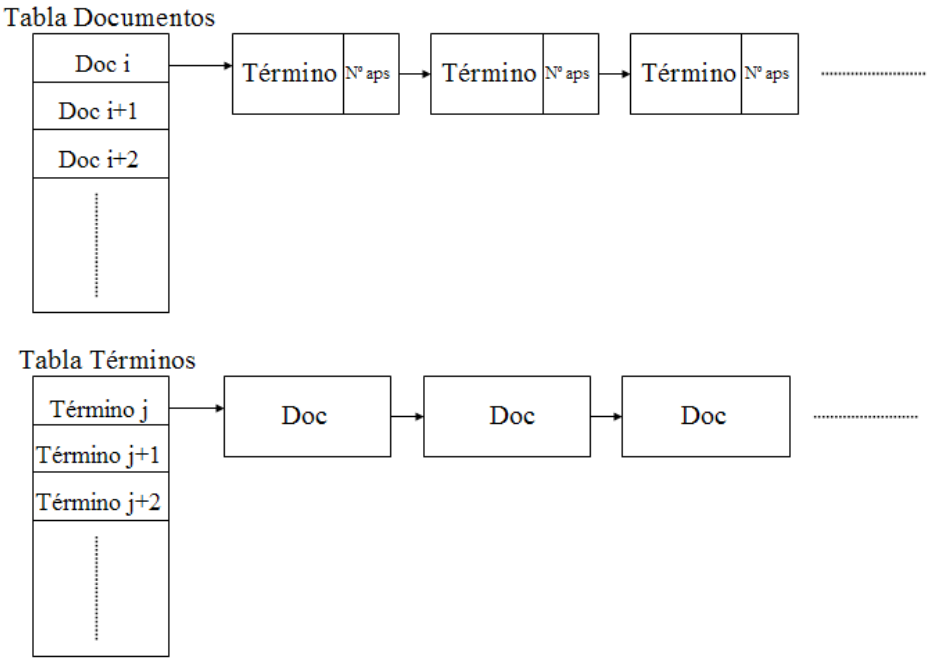

Figura 2.2: Posible implementación para realizar la similitud del coseno

2.3.4. Mejoras de sistemas CBR Textuales

El tratamiento de textos descrito en la sección anterior trata cada palabra individualmente, aunque algunas de ellas están relacionadas, por ejemplo, el sistema descrito anteriormente no distinguiría "pantalón" de "pantalones". Para mejorar estos sistemas se le suele realizar un preprocesamiento al texto (tanto el de la consulta como los de la base de casos). Dicho preprocesamiento puede ser muy variado, aquí mostramos sólo algunos ejemplos:

- El filtrado de palabras vacías (stop-words), que son palabras carentes de significado por sí mismas como los artículos o las preposiciones.

- La extracción de raíces consiste en reducir las palabras a una forma base o raíz que no necesariamente tiene que coincidir con su raíz morfológica. Existen diversos algoritmos para la extracción de raíces como por ejemplo algoritmos de fuerza bruta, estocásticos, de encaje (matching), o de reglas de eliminación de sufijos[6].
- El reconocimiento de sinónimos, por ejemplo "sofá" para "diván". Puede ser beneficioso para el recall[7] pero puede disminuir la precisión[8] si se aplica demasiado agresivamente.[B4]
- La corrección del deletreo para corregir los errores en documentos y consultas.

2.4. Estudio sobre las herramientas empleadas

A continuación analizaremos las herramientas empleadas para el desarrollo de este módulo. Analizaremos el proyecto de SSII de 2005-2006 en el que nos basamos para desarrollar este módulo, el framework para desarrollo de sistemas CBR jColibri, el namespace ScriptSystem para conectar C++ con Java, las características técnicas del juego Javy2, y la herramienta de gestión de configuración Tortoise SVN.

2.4.1. Proyecto de SSII de 2005-2006

La adquisición de conocimiento sobre el proyecto de Sistemas Informáticos de 2005-2006[B1] se realizó mediante la lectura de la memoria de dicho proyecto y el análisis del código, ambos proporcionados por nuestra tutora, Belén Díaz Agudo. Se planteó una reunión con los autores de dicho proyecto pero finalmente no fue necesario puesto que con la memoria y el código se adquirió todo el conocimiento necesario sobre la parte que habíamos de reutilizar.

El proyecto del año pasado consistía en un sistema CBR sobre la máquina virtual de Java. Éste era capaz de proporcionar ayuda al usuario, proponerle ejercicios según su nivel, y diversas funcionalidades más

[6] Ver [B5]-Stemming

[7] recall: Proporción de todos los documentos relevantes en la colección que están en el conjunto resultado.

[8] precisión: Proporción de los documentos en el conjunto resultado que son actualmente relevantes.

en las que no nos detendremos. De todo el proyecto del año pasado solo se reutilizó el módulo de CBR de ayuda textual. Extraer dicho módulo del conjunto del sistema fue relativamente sencillo gracias a la modularidad de dicho sistema.

Módulo CBR de ayuda textual

El módulo CBR de ayuda textual constaba de 3 clases (CBRTextual.java, textualCBR.java, y Ayuda.java), y era prácticamente independiente del resto del sistema, únicamente un método empleaba recursos de otras clases, pero como dicho método no era necesario, se eliminó sin causar mayor problema.

El módulo de ayuda estaba dividido en dos niveles, el nivel superior (CBRTextual) gestionaba el acceso al nivel inferior (textualCBR) según las fases de un sistema CBR textual, que explicaremos más adelante, mientras que el nivel inferior gestionaba el acceso a jColibri mediante las tareas que mandaba ejecutar a jColibri, como se observa en la figura 2.3. La clase Ayuda era una clase de ayuda para almacenar un texto asociado a un identificador.

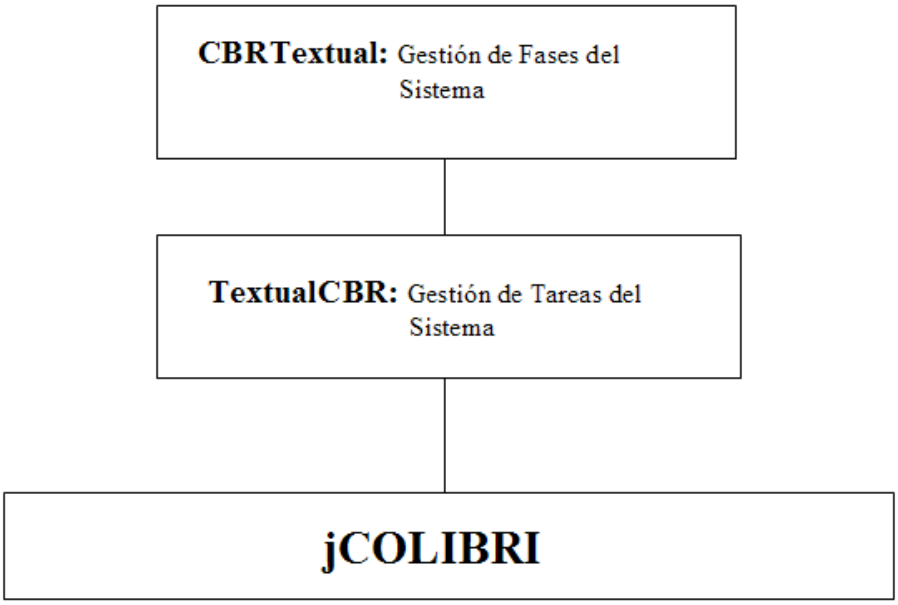

Figura 2.3: Estructura del módulo CBR textual

El objetivo del módulo de ayuda textual de este proyecto tenía por objetivo proporcionar una ayuda fácil e intuitiva al usuario, de forma que éste no tuviese que memorizar todas las instrucciones de la JVM, o incluso describir el funcionamiento de una instrucción y recibir como resultado la instrucción descrita. Nuestro objetivo sigue siendo este mismo, añadiendo que además de ayuda sobre la JVM nuestro sistema deberá proporcionar ayuda sobre el juego Javy2. El módulo de ayuda textual estaba integrado dentro de un sistema propio, con una interfaz propia, mientras que nuestro módulo estará integrado dentro de Javy2, así que no podrá mostrar nada por pantalla, todas las respuestas serán devueltas a Javy2, quien se encargará de mostrarla por pantalla. Al estar integrado en Javy2 nuestro sistema tendrá que tener un tiempo de respuesta bajo, que no pare el funcionamiento del juego, más bajo que el requerido por el módulo de este proyecto. Este sistema sólo dispone de un módulo de ayuda textual y la conexión con el interfaz es sencilla, mientras que nuestro sistema, al constar de dos módulos y estar conectado con Javy2, deberá estructurarse de forma que dicha conexión sea lo más sencilla posible.

La extensión de objetivos y su integración en un entorno interactivo hace que no se pueda usar el módulo obtenido del proyecto del año pasado tal cual, habríamos de adaptarlo a nuestro proyecto.

Las adaptaciones necesarias que tendríamos que realizar serían las siguientes:

1. **Modificación de la estructura de paquetes:** Era necesaria una adaptación de la estructura de paquetes para que la inclusión de este módulo resultase coherente con la estructura general de nuestro proyecto.
2. **Creación de un nivel superior:** Puesto que nuestro proyecto debería ser fácilmente ampliable, la capa superior no debería depender de la implementación concreta de un tipo de ayuda específico, por lo que tuvimos que hacer un nivel superior que cualquier módulo de ayuda pudiese implementar, veremos más adelante la estructura general de nuestro sistema y cómo encaja el módulo CBR de ayuda textual en ella.
3. **Modificación de la clase Ayuda:** Tendríamos que modificar la clase Ayuda para que además de un identificador y un texto

incluyese la similitud con la pregunta efectuada. Esto fue necesario porque nuestro sistema permitiría mostrar la siguiente ayuda en similitud si la mostrada en primer lugar no era satisfactoria.

4. **Modificación del sistema de archivos:** Las rutas de los archivos de configuración que empleaba el módulo CBR de ayuda textual estaba fijada para la estructura de directorios empleada en ese proyecto, habría que cambiarla para que se adecuase a nuestra estructura de directorios.

5. **Modificación de la secuencia de fases:** El módulo CBR de ayuda textual realiza la carga completa de la ayuda en cada consulta repercutiendo en un tiempo de respuesta demasiado alto. Puesto que nuestro sistema estará integrado dentro de Javy2, con unos requisitos de tiempo de respuesta, es necesario reducir dicho tiempo de respuesta. Esto se conseguirá cargando la base de casos al inicio del programa y no en cada consulta.

6. **Correcta actualización de similitudes:** Puesto que en cada consulta se cargaba la base de casos de nuevo, no era necesario actualizar las similitudes en cada consulta, pues estas comenzaban de nuevo. Cuando modificásemos dicho aspecto habría que actualizar correctamente las similitudes en cada consulta.

7. **Inclusión de un umbral:** Se incluyó un umbral para que solo se mostrasen los textos de ayuda con cierta similitud a la consulta realizada.

8. **Eliminación de componentes gráficos:** Puesto que nuestro sistema iba a estar integrado dentro de Javy2 no era admisible la aparición de ciertos componentes gráficos, como barras de progreso, que si tenían sentido en el entorno gráfico para el que se pensó inicialmente este módulo.

Estos son algunos de los problemas detectados mediante la lectura de la memoria y el análisis del código, pero no los únicos. A medida que se fue desarrollando el sistema se encontraron y resolvieron otros problemas de carácter técnico.

2.4.2. jColibri

jColibri es una herramienta desarrollada por GAIA. Es un framework orientado a objetos en Java para la creación de aplicaciones de razona-

miento basado en casos (sistemas CBR).

El conocimiento adquirido sobre jColibri provino de diversas fuentes, del análisis del código facilitado, de la observación del uso de jColibri por parte del proyecto del año pasado, de la página web de jColibri [B2], y de las explicaciones de Belén Díaz Agudo y Juan Antonio Recio, a los que hemos de agradecer su paciencia.

"jColibri ha sido diseñado pensando en conseguir una plataforma de desarrollo de sistemas CBR que sirva de referencia en la comunidad científica. El objetivo es conseguir que el framework crezca y evolucione gracias a las aportaciones de distintos especialistas.
jColibri ha sido construido en torno a la idea básica de separar tareas (tasks) y métodos (methods). Las tareas indican objetivos que el sistema debe alcanzar y guían la ejecución de la aplicación. Los métodos indican como resolver las tareas. Por ejemplo, una idea bastante aceptada es que el ciclo principal de CBR puede descomponerse en cuatro tareas: recuperar los casos mas similares (retrieve), reutilizarlos para resolver el problema (reuse), revisar la solución propuesta (revise) y aprender de la experiencia (retein). Siguiendo esta idea, jColibri modela el ciclo CBR mediante la tarea CBR Task y propone un método asociado CBR Method que la descompone en 4 subtareas: CBR Retrive Task, CBR Reuse Task, CBR Revise Task y CBR Retein Task. A su vez para resolver estas tareas habrá que seleccionar los métodos correspondientes."[B8].

Uso de jColibri por parte del proyecto del año pasado: El proyecto de Sistemas Informáticos del año pasado se servía de jColibri para realizar el módulo CBR de ayuda textual. La forma de utilizar jColibri que empleaban fue la que nos enseñó cómo realizarlo nosotros.

Como se ha explicado anteriormente el módulo CBR de ayuda textual estaba estructurado en dos niveles, siendo el inferior el que directamente trataba con jColibri, fue este nivel el que estudiamos para aprender a usar jColibri.

El módulo consta de una inicialización en la que se inicia el núcleo de jColibri (CBRCore), se adquieren los paquetes necesarios para el núcleo y para la extensión de textos (Textual Extension), se inicia el núcleo, y posteriormente se configura. Es esta configuración la parte

más importante de entender, pues es en ella en la que se crean las tareas a realizar en cada fase, los métodos con los que se resolverán dichas tareas y los parámetros de esos métodos, si fuese necesario alguno.

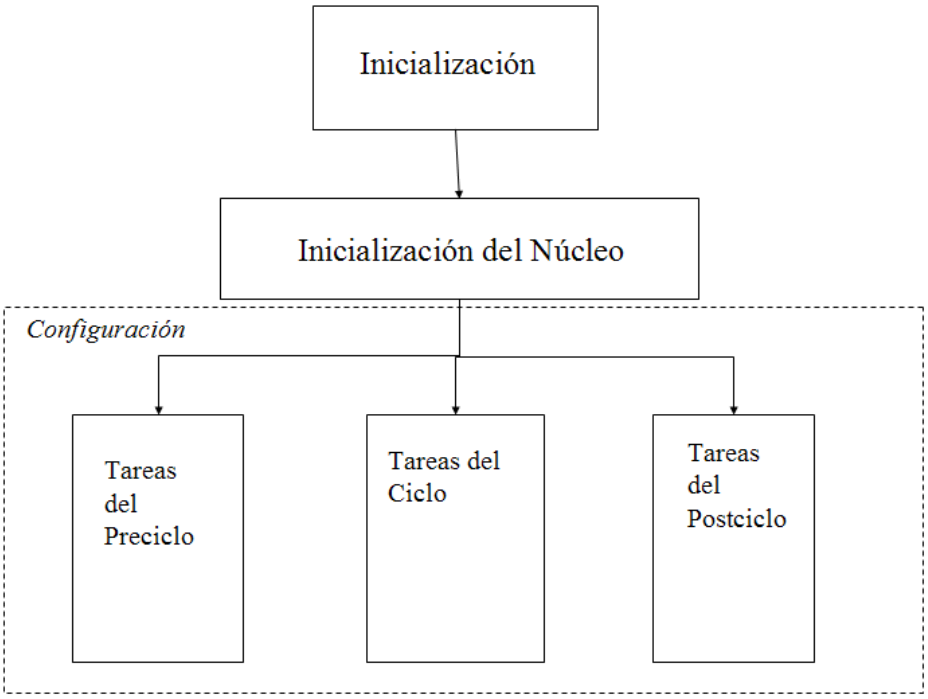

Figura 2.4: Inicialización del módulo CBR de ayuda textual

Adaptación de jColibri: El problema de aprender a usar jColibri por medio del proyecto del año pasado fue que la versión que se empleaba era antigua, y había sido retocada específicamente para ese sistema. Por lo tanto tuvimos que realizar ciertas modificaciones:

1. **Actualización de la pregunta:** Puesto que anteriormente para cada consulta se creaba desde el principio el módulo de jColibri la pregunta era indicada en la construcción de dicho núcleo, eso no nos valía. Para que actualizase correctamente la pregunta se quitaron los atributos específicos de la inicialización de jColibri y se modificó el método de configuración de la pregunta (ConfigureQueryMethod) para que aceptase esta por parámetro. El parámetro pregunta sería pasado por un nuevo método de la clase TextualCBR que habría que invocar desde el nivel de fase antes de ejecutar el ciclo.

2. **Eliminación de los componentes gráficos:** jColibri mostraba barras de progreso para indicar el progreso de cada tarea que realizaba, lo cual no era aceptable si nuestro sistema iba a estar integrado en Javy2. Siguiendo las indicaciones de Juan Antonio Recio modificamos la clase ProgressBar para eliminar este inconveniente.

3. **Generalización de las rutas de los archivos de configuración:** Las rutas de los archivos de configuración en los que se definen tanto el conector a emplear como la estructura de los casos estaban definidas dentro del propio jColibri. Esto se resolvió pasando por parámetro las rutas de dichos archivos a los métodos que lo necesitaban, el método para cargar la base de casos (LoadCaseBaseMethod) y el método para configurar la pregunta (ConfigureQueryMethod).

Como hemos explicado anteriormente cada fase del CBR se define mediante una serie de tareas y métodos, que siguen el modelo de capas de Lenz[B11], ¿pero qué es lo que hacen estas tareas y métodos? A continuación mostramos las tareas y métodos que inicialmente se realizaban en cada fase:

- **Preciclo:** En el preciclo se ejecutan las tareas definidas, que consisten en obtener la base de casos, textos con los que compararemos la pregunta, convertir dichos textos en casos de nuestro sistema CBR y procesarlos de forma que solo se almacenen las palabras representativas del texto, por ejemplo eliminando artículos, pronombres, etc. y extrayendo la raíz de los verbos.

    1. **PreCycle:** *TextualPreCycleMethod*. Esta tarea no realiza ninguna acción específica, está incluida para identificar el inicio del Preciclo.
    2. **Obtain cases task:** *LoadCaseBaseMethod*. Esta tarea se encarga de abrir la conexión con el CBR usando el conector indicado por parámetro y carga todos los casos de la base de casos.
    3. **Select working cases task:** *SelectAllCasesMethod*. Esta tarea se encarga de especificar con que casos de la base se va a trabajar, en nuestro caso se seleccionan todos, lo que viene indicado por el método de resolución de la tarea.
    4. **Cases Textual Process task:** *TextualCBRPreProcessMethod*. Indica que va a empezar el preprocesamiento de los textos de la base de casos.

5. **Keyword Layer:** *KeywordLayerMethod*. Tarea de descomposición, actualmente no realiza ninguna acción.
6. **Words Filter:** *WordsFilterMethod*. Filtra las palabras que se pueden quitar (stop-words) y los caracteres especiales. Las palabras que se pueden quitar son palabras habituales carentes de significado por si mismas, como por ejemplo los artículos.
7. **Stemmer:** *StemmerMethod*. Esta tarea realiza el algoritmo Stemmer. Éste consiste en reducir las palabras a su raíz de forma que se puedan comparar por ejemplo verbos en diferentes tiempos o personas. Por ejemplo reduciría *comprar* a *compr*, igual que haría con *comprando* o *compraba*, de forma que se pueda comparar correctamente *comprar*, *comprando* o *compraba*, puesto que las tres se refieren al mismo concepto.
8. **Part-of-Speech:** *PartofSpeechMethod*. Esta tarea se encarga de catalogar las palabras de forma que podamos saber si son pronombres, adjetivos comparativos o superlativos, verbos en su forma base,etc.
9. **Extract Names:** *ExtractNamesMethod*. Esta tarea selecciona los nombres principales del texto.
10. **Phrase Layer:** *ExtractPhrasesMethod*. Esta tarea extrae frases usando expresiones regulares.
11. **Feature Value Layer:** *ExtractFeaturesMethod*. Esta tarea extrae características usando expresiones regulares.
12. **Domain Structure Layer:** *DomainTopicClassifierMethod*. Esta tarea asocia un tema al texto usando las frases y características obtenidas anteriormente como condiciones.
13. **Information Extraction Layer:** *BasicIEMethod*. Esta tarea extrae información del texto y la almacena en casos individuales.

- **Ciclo:** En el ciclo se aplican las mismas fases a la query para poder compararla con los casos que se crearon en el preciclo y que están en memoria.

  1. **CBR Cycle:** *CBRMethod*. Esta tarea no realiza ninguna acción, está aquí para indicar el inicio del ciclo.
  2. **Obtain query task:** *ConfigureQueryMethod*. Esta tarea lee la estructura de los casos de un archivo y almacena la pregunta en la estructura definida en el archivo.

3. **Retrieve Task:** *TextualCBRRetrieveMethod*. Tarea de descomposición,actualmente no realiza ninguna acción.
4. **Query Textual Process task:** *TextualCBRPreProcessMethod*. Indica que va a empezar el preprocesamiento de los textos del caso pregunta.
5. **Keyword Layer:** *KeywordLayerMethod*.
6. **Words Filter:** *WordsFilterMethod*.
7. **Stemmer:** *StemmerMethod*.
8. **Part-of-Speech:** *PartofSpeechMethod*.
9. **Extract Names:** *ExtractNamesMethod*.
10. **Phrase Layer:** *ExtractPhrasesMethod*.
11. **Feature Value Layer:** *ExtractFeaturesMethod*.
12. **Domain Structure Layer:** *DomainTopicClassifierMethod*.
13. **Information Extraction Layer:** *BasicIEMethod*.
14. **Select working cases task:** *SelectAllCasesMethod*. Esta tarea selecciona los casos con los que se va a trabajar, en nuestro caso con todos como viene indicado por el método de resolución.
15. **Text Relation Process task:** *TextRelationsMethod*. Esta tarea no realiza ninguna acción, esta aquí para indicar el comienzo de la comparación entre textos.
16. **Thesaurus Layer:** *WordNetRelationsMethod*. Esta tarea relaciona palabras de la pregunta con palabras de los textos de la base de casos utilizando WordNet.
17. **Glossary Layer:** *GlossaryRelationsMethod*. Esta tarea relaciona palabras de la pregunta con palabras de los textos de la base de casos utilizando un glosario específico del dominio.
18. **Compute similarity task:** *NumericSimComputationMethod*. Esta tarea calcula la similitud entre la pregunta con los casos con los que trabajamos de la base de casos.
19. **Select best task:** *SelectSomeCasesMethod*. Esta tarea selecciona los mejores casos(los de mayor similitud). En nuestro caso se seleccionan todos ordenados por similitud puesto que la selección se realiza posteriormente en la creación del vector de resultados.

- **Postciclo:** Esta fase se encarga del cierre de los conectores abiertos.

1. **PostCycle:** *CloseConnectorMethod*. Esta tarea se encarga del cierre de los conectores abiertos.

Como se puede observar existían tareas que no realizaban ninguna acción concreta, y otras que no eran realmente necesarias para nuestro sistema. Puesto que jColibri tiene una potencia que nosotros solo íbamos a emplear en parte, descartando por ejemplo la asignación de características (features) a los textos, eliminamos ciertas tareas que no influían en el resultado, solo consumían tiempo de ejecución.

2.4.3. ScriptSystem

El conocimiento adquirido sobre este apartado fue facilitado mediante el documento *"ScriptManager.pdf" de Marco Antonio Gómez Martín* así como por sus respuestas a las dudas generadas tras su lectura.

Introducción:

ScriptSystem es un *namespace* que contiene todas las clases necesarias para facilitar la comunicación entre C++ y Java, encubriendo el uso del interfaz JNI[9].

"Aunque Javy inicialmente está implementado en C++, existe una parte reducida que está implementada en Java, y que permite acceso a la funcionalidad, por ejemplo, proporcionada por jColibri, el framework de CBR desarrollado por GAIA.
Para integrar Java en C++, se utiliza JNI, el interfaz Java-C utilizado por las aplicaciones Java para poder llamar a C++. Una explicación sobre JNI (fuera del contexto de Javy) puede encontrarse en `http://gaia.fdi.ucm.es/people/marcoa/development.html` *....*
... Todas las clases relacionadas con la gestión de las clases Java se encuentran en el directorio .\src\ScriptSystem, y bajo el namespace ScriptSystem[10]*.*
El diseño se ha hecho pensando en intentar facilitar lo máximo posible el uso de clases/objetos Java desde C++, aunque no elimina la necesidad de implementar los "stubs" típicos de estos casos (existen librerías que sí lo hacen, como Jace). "[B9].

[9] JNI:Java Native Interface
[10] Debemos exceptuar un namespace JavaIntfz que oculta JNI.

ScriptManager

Esta clase se ocupa de gestionar todo lo relativo a la máquina virtual de Java, se encarga de su carga y liberación, de indicar el classpath que se utilizará, de cargar clases, y de crear objetos a partir del nombre de la clase. Como el gestor debe ser único se ha implementado como un *Singleton*.

ScriptClass

Esta clase representa una clase de la máquina virtual de Java. Los objetos de esta clase son creados mediante el uso del ScriptManager, que devolverá un puntero a un objeto de esta clase. Puede crear objetos Java, aunque se uso principal es la invocación de métodos estáticos.

ScriptObject

Esta clase simboliza un objeto de la máquina virtual de Java. Puede ser creado mediante el ScriptManager o el objeto de la clase a la que pertenece ScriptClass. Su uso es el de la invocación de los métodos no estáticos de la clase. A diferencia de las anteriores clases la liberación de memoria corre por parte del usuario, que deberá borrar el puntero a este objeto cuando haya finalizado su uso.

Hay que remarcar el tratamiento especial que se les da a los Strings de Java mediante la creación de una clase StringScript que hereda de StringObject, al que se le añade un método para devolver la cadena en C++ del correspondiente String de Java. También existe un método para la creación de un String a partir de una cadena en C++, aunque pueden existir problemas debido a que en C++ la codificación es ASCII mientras que en Java es Unicode.

Invocación de métodos

Como hemos visto antes los métodos estáticos se invocan desde un objeto del tipo ScriptClass mientras que los métodos no estáticos o de objeto se invocan desde un objeto del tipo ScriptObject. La invocación de ambos tipos de métodos es análoga, por lo que solo explicaremos un

tipo, los métodos de objeto.

Existen varios métodos a invocar según el tipo devuelto por el método java que queremos invocar, pero en todos ellos el primer parámetro es el nombre del método Java, el segundo es la signatura del método, y los siguientes son una serie de parámetros variable y opcional que se corresponden con los parámetros que necesita el método Java. La signatura es una codificación del tipado del método Java en la que por ejemplo el tipo "int" se corresponde con la letra "I". A continuación vemos un ejemplo de uso:

```
obj->InvokeVoid("setFecha", "(III)V", 2, 3, 1989);
```

Soporte para hebras

"La máquina virtual de Java permite que distintas hebras se ejecuten sobre ella. Es más, también permite que varias hebras de C++ invoquen a métodos suyos simultáneamente.
Lo normal es utilizar JNI de tal forma que la máquina virtual tenga el control completo de la aplicación, y de vez en cuando llame a métodos nativos de C++. Eso puede provocar que, si en Java se han creado hebras, se estén ejecutando simultáneamente varios métodos nativos implementados en C++.
En nuestro caso, la aproximación es a la inversa, por lo que el tratamiento de hebras también. Más concretamente, estamos utilizando Java desde C++, es decir, en C++ hemos creado una JVM, y llamamos a métodos de sus objetos.
Pues bien, ese programa de C++ puede crear distintas "hebras C++", y querer llamar a métodos Java..."[B9].

Pese a esta capacidad de generar diferentes "hebras C++" en nuestro caso optamos por generar las hebras desde Java, pues nos pareció una aproximación más sencilla e igualmente efectiva.

¿Por qué necesitábamos emplear más de una hebra en nuestro programa?. Pues bien, esto se debió a que el tiempo de inicialización de nuestro sistema era relativamente alto, y al inicializarse al iniciar Javy2 éste iba a estar parado hasta que terminase la inicialización de nuestro sistema. Para evitar este problema optamos por que en el método

Java de inicialización se crease una segunda hebra que realizase efectivamente la inicialización en segundo plano mientras que la primera hebra devolvía el control a Javy2 para que pudiese continuar la carga del juego. Podemos verlo gráficamente en la figura 2.5.

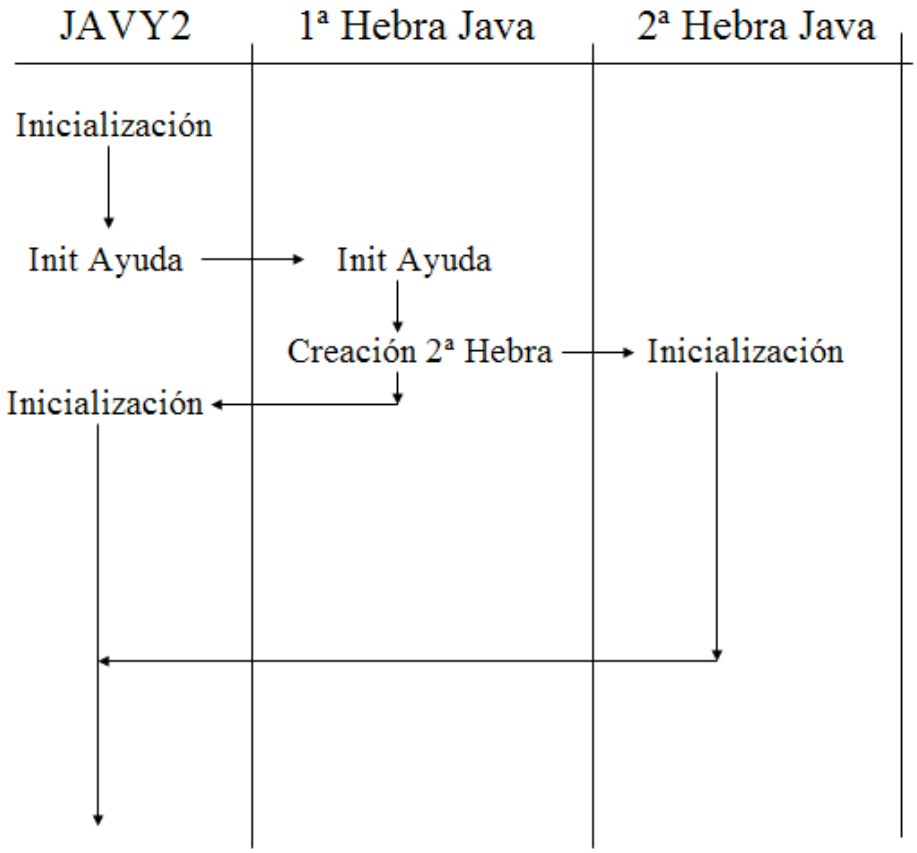

Figura 2.5: Inicialización en segundo plano

2.4.4. Javy2

A continuación exponemos el conocimiento adquirido sobre Javy2, dicho conocimiento se referirá exclusivamente a la parte técnica de Javy2[B10], pues el conocimiento adquirido sobre la metáfora de Javy2 se explica más adelante en la sección 3.3.

Estructura de directorios de Javy2:

La estructura de directorios es la siguiente [B10]:

- *Docs: ficheros de documentación. Contiene por ejemplo el código LaTeX del documento que estás leyendo.*
- *Exes: directorio donde se genera el ejecutable en la compilación. Contiene también todos los ficheros de arte necesarios.*
- *JavyAux: contiene proyectos auxiliares que se necesitan o bien para poder compilar Javy 2 o bien se necesitan para generar algunos de los ficheros de datos (mapas principalmente).*
- *Libs: librerías necesarias para la compilación.*
- *Projects: proyectos de compilación. También en ese directorio se generan los ficheros intermedios resultados de la compilación (obj).*
- *Src: directorio con el código fuente.*

Mapas de Javy2:

"Los ficheros que en el SVN están almacenados como "fuente" y que necesitan ser exportados son los mapas. *Los ficheros de mapas son los que contienen la arquitectura del entorno, paredes, localización inicial del personaje, manadas, puertas, etc., y se almacenan en* `.\Exes\Maps\*.rmf`.
Para abrirlos, se necesita el programa editor, llamado Worldcraft, creado por Valve para crear los mapas de su famoso juego Half-Life." [B10]

La instalación del WorldCraft es sencilla mediante su instalador, la configuración requiere un poco más de trabajo. En *Game Configurations* hay que añadir un nuevo juego (Javy) y asignarle el fichero de datos (`.\Exes\Maps\javy.fgd`). También hay que añadir el archivo de texturas (halflife.wad) en la configuración de *Textures*.

Una vez configurado solo hay que exportar el mapa en .rmf a .map. Tras esto desde la consola de comandos y en el directorio `.\Exes\Maps` se ejecuta la línea:

```
> compilamapa <nombre mapa sin extension>
```

Una vez compilados los mapas ya se puede comenzar la compilación de los proyectos.

Requisitos de Javy2:

Antes de comenzar hay que destacar que para poder compilar y ejecutar Javy2 es necesario tener instalado Microsoft Visual Studio.NET 2003, y las `SDK de DirectX 9`. También es necesario tener instalado el `JDK de Java 1.4` o posterior. Para configurar Microsoft Visual Studio.NET 2003 para que pueda emplear el JDK de Java se debe hacer:

"Para que se encuentren los ficheros de cabecera necesarios, debe estar incluido en el PATH del preprocesador el directorio `$JDKROOT$\include` y `$JDKROOT$\include\win32`[11]*. Para configurar el Visual Studio, en el Menú "Herramientas", "Opciones", en la "carpeta" llamada "Directorios", añadir en los "Directorios de VC++", los dos anteriores, en el apartado de "Archivos de inclusión".*

El proyecto de Javy, además, hace uso de un .lib del JDK para poder utilizar la DLL de la máquina virtual. Para que Visual C++ la encuentre, también habrá que decirle dónde está. En la instalación del JDK, se crea en `$JDKROOT$\lib`*. Habrá que añadir ese directorio en el apartado "Archivos de biblioteca", accesible desde el mismo cuadro de diálogo que el anterior.*

Con todo lo anterior, se conseguirá que Javy se genere correctamente, pero no necesariamente que se ejecute... Para eso, se tendrán que cumplir dos condiciones:

- *Que la DLL de Java esté en el PATH. Si se tiene el JRE instalado, meter en el PATH* `$JREROOT$/bin/client`[12].
- *Que el CLASSPATH apunte donde tenga que apuntar que estén las clases de Java. Normalmente,* `$JREROOT$/lib`.

Para hacer globales estos cambios en el entorno, en el Panel de Control - Sistema - Opciones avanzadas - Variables de entorno."[B10]

[11] `$JDKROOT$` apunta al directorio de instalación del JDK

[12] Si no se a instalado explícitamente el JRE, éste está en $JDKROOT$/jre

Proyectos Auxiliares:

Javy2 consta de una serie de proyectos auxiliares que hay que compilar antes de poder compilarlo. Casi todos ellos se compilan de la misma manera, desde Visual Studio seleccionar Generar→Generación por lotes... →Seleccionar todo + Generar. Dichos proyectos son los siguientes:

- **BSPFile:** *"Este proyecto genera la librería necesaria para leer ficheros .BSP (que contienen los mapas compilados anteriormente). El proyecto está hecho de tal forma que es capaz de generar tanto librerías estáticas (.lib) como dinámicas (.dll)."*[B10].
- **GVME:** *"Este es el código del editor de mapas originalmente desarrollado por los alumnos del Master de Videojuegos 2005/06 (segunda edición). No requiere compilación, y es raro que se necesite utilizar."*[B10].
- **Nebula2:** *"Contiene el código del motor gráfico utilizado en Javy 2. Actualmente, utiliza la versión liberada por RadonLabs en su Release de 2005, con ligeras modificaciones.*
  Para compilarlo:
  1. *Ejecutar* `./JavyAux/Nebula2/update.exe`. *Seleccionar vstudio71 en "generator", y "nebula2javy" en Workspaces. Dar al boton Run y luego a Close.*
  2. *Abrir el proyecto* `./JavyAux/Nebula2/build/vstudio71/nebula2javy.sln` *que se habrá creado en el paso anterior, y compilar todos sus proyectos, utilizando la generación por lotes ya explicada antes."*[B10].
- **opcode:** *"Librería de detección de colisiones. En Javy 1 la detección estaba implementada, a través de una fachada, por Nebula 1. En Javy 2 se eliminó la necesidad de Nebula 1 utilizando directamente opcode."*[B10].
- **UnitTest++:** *Una parte de Javy 2 está desarrollado siguiendo la metodología de desarrollo dirigido por Tests (en inglés, Test Driven Development o TDD). Para ello, se utiliza una librería para ejecutar los tests, llamada UnitTest++, que se almacena en el directorio* `.\JavyAux\UnitTest++`*."*[B10].

Compilación de Javy2:

"El último y esperado paso es compilar la solución de Javy 2. Está en el directorio raiz, llamado Javy2.sln. Se puede compilar tanto en modo depuración como en modo Release."[B10].

2.4.5. Tortoise SVN

Puesto que debíamos integrar nuestro proyecto dentro de Javy2 se nos facilitó acceso al medio de gestión de configuración empleado en Javy2, que recientemente se había cambiado de CVS a Subversion, un sistema de control de versiones más avanzado que el CVS tradicional. Para la plataforma Windows el cliente de Subversion es TortoiseSVN[13], que se integra con el Explorer.

"Si ya está instalado, para bajar los ficheros de Javy2, crear un directorio, y desde el Explorador, abrir el menú contextual (botón derecho del ratón) y elegir la opción "SVN Checkout...". En el cuadro de diálogo que se abre, indicar como URL del almacén ("URL of repository"): `svn://javy.sip.ucm.es/javy/Javy2/trunk` *que viene a indicar que el almacén está en la máquina* `javy.sip.ucm.es`*, y el almacén es Javy. Dentro de este almacén, se bajará la rama principal (trunk) del proyecto Javy2. El cliente de Subversion pedirá la identificación del usuario (nombre de usuario y contraseña), que habrá tenido que ser facilitada por el grupo de desarrollo principal."*[B10].

2.5. Desarrollo

Como vimos en las secciones 2.1 y 2.4.1 nuestro sistema debe proporcionar una ayuda ágil e intuitiva al jugador de Javy2, debe ser altamente adaptable, debe integrarse de la manera más sencilla posible, debe tener un tiempo de respuesta alto, y por lo tanto debe tener una estructura claramente dividida en módulos. Estos problemas fueron resueltos mediante la estructura del sistema que explicamos a continuación.

2.5.1. Estructura General

El sistema está formado por una clase de acceso, la clase Help, que se compone de los diferentes módulos que forman el sistema de búsqueda,

[13] `http://tortoisesvn.tigris.org/`

de forma que al buscar la respuesta a una pregunta busca en todos los módulos y selecciona la mejor respuesta de todos ellos. De hecho está formado por interfaces de dichos módulos de forma que puedan usarse varias implementaciones de los módulos, por ejemplo una implementación por idioma. El sistema posee además varias clases de apoyo para estructuras de datos, la ejecución concurrente, o por ejemplo para mostrar un mensaje según el idioma elegido.

¿Por qué se han usado diferentes módulos en vez de uno solo? La elección del uso de diferentes módulos, uno por ámbito de búsqueda del sistema[14], viene motivada por la facilidad de ampliación, la facilidad de modificación y por mejoras de eficiencia. Facilidad de ampliación porque si se quiere añadir un nuevo ámbito de búsqueda no haría falta tocar los existentes. Facilidad de modificación porque si se quiere cambiar la búsqueda sobre un ámbito sólo sería necesario modificar la implementación de dicho módulo. Por eficiencia, por la posibilidad de realizar las búsquedas en los diferentes módulos en paralelo, lo que con la implantación de procesadores multicore irá poco a poco cobrando importancia.

Cada módulo está formado por el interfaz de dicho módulo, una clase que implementa dicho interfaz. Esta clase está formada por un vector de resultados y por una clase que implementa el acceso a alto nivel de jColibri. La clase de acceso a jColibri controla la ejecución de las fases de jColibri[15] mediante la invocación de métodos de una clase de acceso a bajo nivel de jColibri. Esta clase tiene un objeto del tipo CBRCore, que es propiamente jColibri, y diversos atributos necesarios para trabajar con jColibri, como por ejemplo la lista de tareas del preciclo, ciclo, y postciclo[15], y se encarga de la configuración de jColibri así como de ordenar la ejecución de tareas, crear el vector de resultados, etc.

Podemos ver esta estructura en la figura 2.6.

[14]Por ámbito de búsqueda nos referimos a dominio de conocimiento. En nuestro caso actualmente existen dos, la ayuda sobre las instrucciones de la JVM, y sobre el juego Javy2.

[15]Ver sección 2.4.2

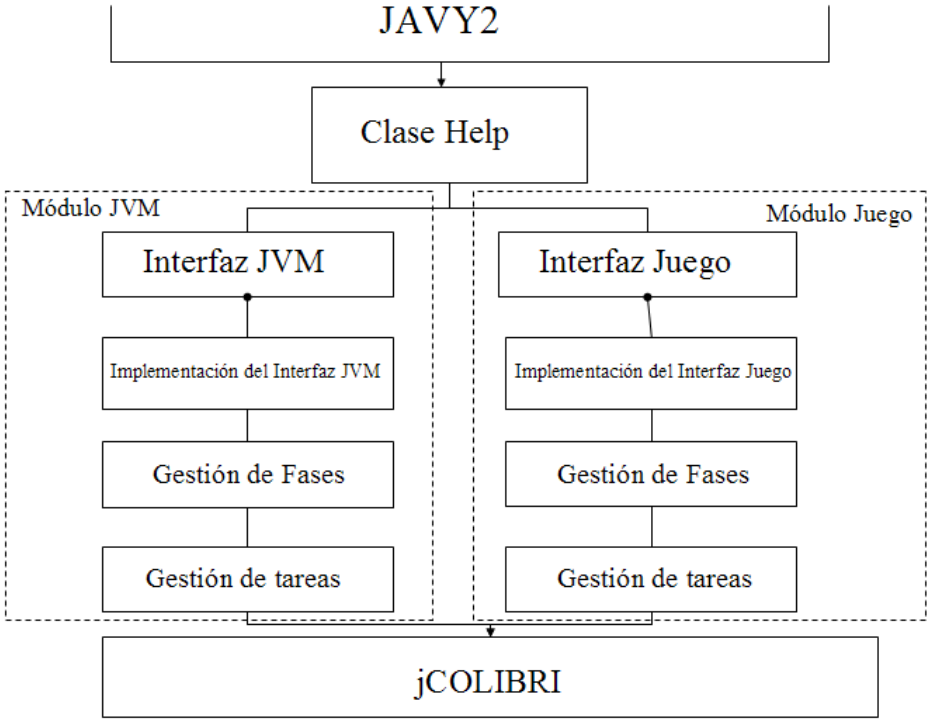

Figura 2.6: Estructura de clases del sistema

2.5.2. La clase Help

La clase Help es la que proporciona un punto de acceso al sistema, es decir, es la clase que será invocada por Javy2 mediante el método `public String help(String query)`. Dicho método comprobará si la pregunta efectuada es una frase clave para activar el *"more like this"*, en caso afirmativo devolverá la siguiente respuesta más similar de cualquiera de sus módulos. Si no es una frase clave solicitará a los diferentes módulos que efectúen una búsqueda sobre jColibri, esta búsqueda genera en cada módulo un array de respuestas ordenadas por similitud. La clase Help solicitará la respuesta más similar de cada módulo y se quedará con la mejor, que será la que devuelva a Javy2.

Podemos ver la secuencia de acción de una pregunta en la figura 2.7.

Esta clase es la que se encarga de efectuar la inicialización en segundo plano, como vimos en la figura 2.5, realizando la inicialización de todos sus módulos en una segunda hebra.

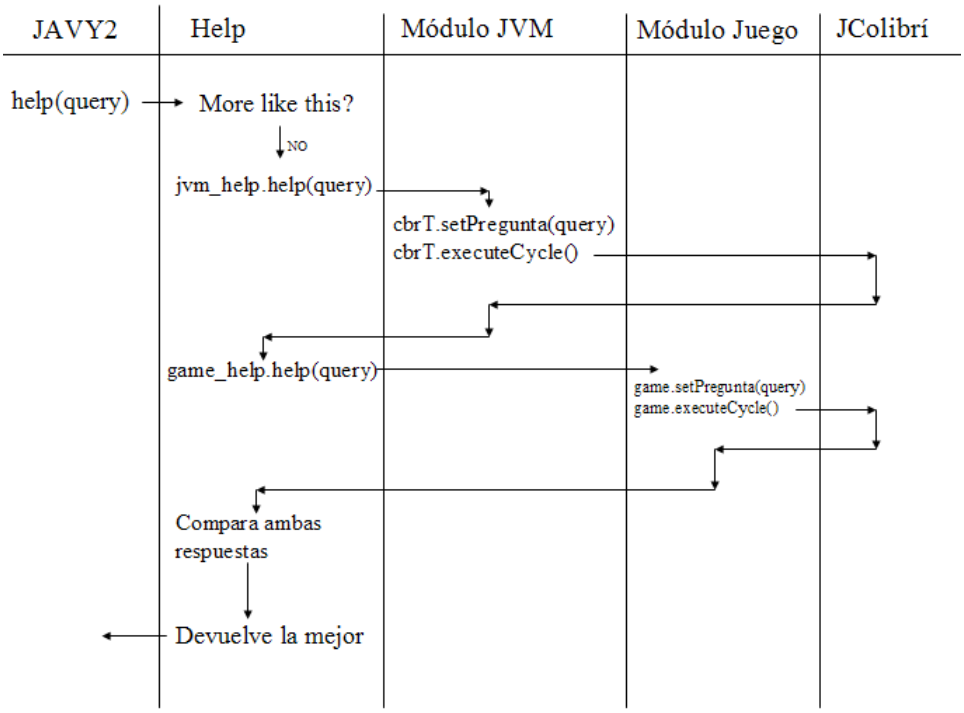

Figura 2.7: Secuencia de ejecución al efectuar una pregunta

También podemos ver la forma en que se actualizan los datos tras una consulta *"more like this"* en la figura 2.8. En dicha figura se muestra la situación inicial de los índices (en azul), tras comparar las respuestas apuntadas por los índices devuelve la que tiene mayor similitud (en verde), y actualiza el índice de la respuesta devuelta (en rojo).

2.5.3. Módulos de ayuda

Cada módulo consta de una serie de clases, un interfaz que establece una forma genérica de acceso al módulo, una o varias clases que implementen dicho interfaz, una clase de alta abstracción de acceso a jColibri y una clase de baja abstracción de acceso a jColibri. Todas estas clases, excepto el interfaz, tienen una relación de inclusión entre ellas, es decir, la clase que implementa el interfaz incluye un objeto de la clase de alta abstracción de acceso a jColibri, y esta clase a su vez

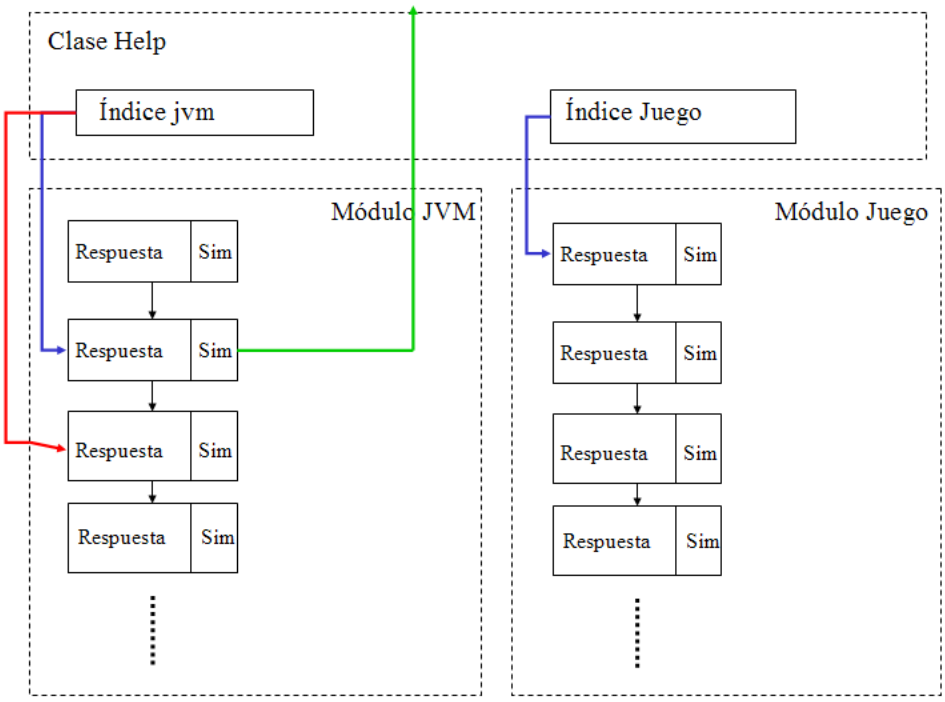

Figura 2.8: Funcionamiento del "more like this"

incluye un objeto de la clase de baja abstracción de acceso a jColibri.

Existe un interfaz por cada módulo, estos interfaces deberán tener una serie de métodos que permitan realizar una búsqueda, obtener el elemento (y su similitud con la pregunta) de una posición del vector de resultados, así como un método para el cierre de los conectores empleados.

Las clases que implementan los interfaces deben lógicamente implementar sus métodos. Para ello poseen un vector donde almacenan el resultado de cada búsqueda (como se puede ver en la figura 2.8). Cada vez que se ejecuta una búsqueda el vector de resultados se limpia y se actualiza con el vector devuelto por la clase que conecta con jColibri a nivel de fase, que veremos más adelante.

2.5.4. Conexión con jColibri

Como vimos en la figura 2.3 la conexión con jColibri se realiza en dos niveles, a nivel de fase y a nivel de tarea. A continuación detallaremos ambos niveles.

Abstracción a nivel de fase

Esta clase controla el acceso a jColibri mediante el control de la ejecución de sus diferentes fases.

En la construcción de un objeto de esta clase se realiza la inicialización de la clase de acceso a jColibri a nivel de tarea así como la fase del preciclo, que se encarga de la carga de la base de casos.

Al solicitar una búsqueda textual se le dice a la clase de acceso a jColibri a nivel de tarea que actualice jColibri con la nueva pregunta, que ejecute la fase de ciclo, que efectúa propiamente la búsqueda, y que cree el vector ordenado de resultados, que es lo que devuelve.

El método de cierre de conectores se encarga de mandar ejecutar la fase de postciclo.

Abstracción a nivel de tarea

Esta clase se encarga del acceso a jColibri mediante la definición de tareas y asignación de métodos para ejecutar dichas tareas. Contiene numerosos atributos para el acceso a jColibri pero caben destacar las listas de tareas del preciclo, ciclo, y postciclo. También posee numerosas constantes de clase donde están definidas las rutas de los archivos de la configuración de jColibri para este módulo.

En su método de inicialización se encarga de iniciar jColibri y de asignar que tareas y que métodos se ejecutaran en cada fase, mediante el método de crear una lista de tareas para cada fase, que luego mandará ejecutar cuando se le ordene desde la clase de acceso a jColibri a

nivel de fase[16].

Para actualizar la pregunta obtiene de jColibri el método que configura la pregunta (*ConfigureQueryMethod*), y le actualiza los parámetros con los nuevos valores.

Para crear el vector de resultados obtiene el resultado de la búsqueda de jColibri y va extrayendo el texto de respuesta y su similitud de los casos devueltos por jColibri(*CBRCase*) con alguna similitud con la pregunta hasta completar un máximo de 20 resultados.

Conectores

Es necesario la definición de un conector con jColibri para cada módulo. La clase de baja abstracción de acceso a jColibri le pasa a este la ruta donde está el archivo de configuración del conector, uno de cuyos parámetros especifica la clase que implementa dicho conector. Todos los conectores deben implementar el interfaz Connector definido en `jcolibri.cbrcase.Connector`. Su función consiste en leer los textos de la base de casos y almacenarlos como casos de jColibri con la estructura definida en el archivo de configuración correspondiente. También se encarga del acceso a los casos y de almacenar casos si fuese necesario.

2.5.5. Clases de apoyo

Las clases de apoyo son clases necesarias para el funcionamiento del sistema pero que no entran dentro de ningún módulo.

- **La clase Messages:** Esta clase consta únicamente de métodos estáticos que reciben por parámetro un idioma y devuelven un mensaje en dicho idioma. También consta de métodos que reciben una frase y un idioma y comprueba si dicha frase cumple cierta condición en ese idioma, como por ejemplo si una frase es una frase clave para ejecutar el *"more like this"*.
- **La clase Ayuda:** Esta clase implementa el TAD utilizado para almacenar las respuestas en el vector de resultados. Consta de

[16]Como vimos en la sección 2.4.2

tres atributos, un String con el texto de la respuesta, un entero con el identificador, y un float con la similitud de la respuesta.

- **La clase Hebra_Inicializacion:** Esta clase implementa el interfaz Runnable de Java y define la acción a ejecutar por la hebra secundaria de inicialización. Sólo dispone de un método, el método *run* en el que se indica que debe realizar la inicialización de los módulos actualizando correctamente las variables de control tomando las correspondientes precauciones por medio del cerrojo que contiene la clase Help.

2.5.6. Conexión con Javy2

La clase ScriptManager abstrae el uso del interfaz JNI, pero no evita que se tengan que implementar cada método de las clases java que empleemos. La clase CHelp implementa el acceso a la clase Help de nuestro programa en java, de forma que, cuando invoquemos un método de la clase CHelp, estaremos invocando el correspondiente método de la clase Help de java. De esta forma conseguimos abstraer completamente la implementación de la parte de nuestro programa hecha en java.

La clase CHelp está implementada como un singleton para garantizar la existencia de un único ejemplar de esta clase.

Para usar la clase CHelp lo primero que tenemos que hacer es inicializar la clase CScriptManager, puesto que si no está creada la máquina virtual la inicialización de la clase CHelp fallará (devolverá falso).

Tras inicializar el ScriptManager debemos inicializar la clase CHelp con el método correspondiente. Una vez hecho esto podemos invocar los métodos askForHelp o languageChange según lo que queramos hacer. Puesto que la clase Help por defecto se crea en inglés si quisiésemos empezar en otro idioma deberíamos inicializar la clase CHelp y luego hacer el cambio de idioma. Esto es debido a que ScriptManager no acepta constructores con parámetros.

Recordemos que la clase Help se inicializaba con una hebra en segundo plano, por lo que si se invocan los métodos mientras se está inicializando la respuesta a la pregunta será un mensaje informando de que

la inicialización todavía no ha concluido.

La inicialización de la clase ScriptManager recibe como parámetros las rutas que conforman el classpath que vamos a pasar a la máquina virtual de java.

La creación e inicialización es un procedimiento costoso, por ello se implementó en una hebra secundaria. Para aprovechar esta característica se inicializa al principio de la inicialización del juego. En concreto la inicialización se realiza en el archivo JavyApp.cpp en el método AppNebula::OnInitialize. La liberación de recursos se hace en el mismo archivo JavyApp.cpp en el método AppNebula::OnTerminate.

El sistema de ayuda debe proporcionar respuesta a las preguntas que el usuario introduce por pantalla, por ello la invocación del método askForHelp está en el método que controla la acción del terminal y se invoca una vez comprobado que lo introducido por pantalla no es un comando de ejecución. Esto se hace en el archivo GUITerminal.cpp en el método CGUITerminal::processCommand. Podemos ver como se muestra la ayuda dentro del juego en la figura 2.9.

Es necesario para el funcionamiento del sistema de ayuda añadir las rutas de las librerías empleadas por el sistema en el classpath. Como ya hemos visto el método de inicialización del ScriptManager permite pasar el classpath por parámetro. Puesto que el classpath forma parte de la configuración de la aplicación este se situó en el archivo config.xml que contiene todos los datos de configuración del juego.

El documento de configuración lo lee la clase CConfigFile que almacena las rutas en una tabla hash con clave numérica (la primera ruta es la 0, la segunda es la 1 y así sucesivamente). Esta clase dispone de métodos para saber cuantas rutas forman el classpath y para devolver la ruta de la posición i. De esta forma podemos obtener el array de rutas pasado al método de inicialización del ScriptManager.

Figura 2.9: Obtención de ayuda en Javy2

2.6. Creación de textos de ayuda

Como hemos visto en secciones anteriores cada pregunta realizada al sistema se compara con una base de casos devolviendo los más similares. Dicha base de casos está formada por varios textos, pero, ¿de dónde se han obtenido los textos?.

Los textos que forman la base de casos del módulo de ayuda sobre la máquina virtual de Java eran parte del proyecto de Sistemas Informáticos ([B1]), por lo que no tuvimos que realizarlos nosotros. Estos textos habían sido extraídos de la documentación de la JVM y posteriormente analizados para incluirlos en la base de casos.

Los textos que forman la base de casos del módulo de ayuda sobre el juego tuvimos que crearlos desde cero, para ello tuvimos que adquirir mucho conocimiento sobre el juego (ver más adelante la sección 3.3). Puesto que al mismo tiempo que teníamos que crear estos textos teníamos que crear una ontología sobre el juego decidimos que por cada concepto que definiésemos en la ontología crearíamos un nuevo texto de ayuda. Esto fue vital para organizarnos y posteriormente faci-

litaría la comparación entre ambas ayudas, la sintáctica y la semántica.

El juego Javy2 está en permanente evolución, por lo que la adquisición de conocimiento, y su conversión en textos de ayuda fue complicada, puesto que lo que un día teníamos claro, al día siguiente se podría haber decidido cambiarlo. No obstante poco a poco se fueron perfilando una serie de conceptos a definir.

Puesto que habíamos creado el sistema, sabíamos la forma de introducir textos para que nuestro sistema funcionase mejor, esto es bueno de cara a la creación de textos puesto que los optimizamos para nuestro sistema, pero malo de cara a la evaluación, puesto que conociendo los textos sabríamos condicionar las preguntas para que saliese la respuesta correcta, lo cual no refleja el funcionamiento real de nuestro sistema. Como veremos en el capítulo 4 esto se resolvió realizando una evaluación por parte de personas ajenas al desarrollo de nuestro sistema.

Puesto que Javy2 permanece en constante evolución, en el futuro será necesario actualizar la base de casos, a continuación detallamos los pasos a seguir para añadir nuevos textos:

1. Lo primero que tenemos que hacer es analizar la información que queremos introducir. Se debe comprobar si ya está introducida en la base de textos o si podemos completar alguno de los textos para que incluya la información que deseamos.

   En caso de que ya esté debemos plantearnos porque la información no aparece en respuesta a nuestra pregunta, debemos analizar las diferentes preguntas que se harían para acceder a dicha información y ver si las palabras que forman la pregunta están incluidas en la respuesta (al menos alguna).

   Si la información no está pero se puede añadir a un texto debemos preguntarnos si al realizar las preguntas para acceder a la información que se desea introducir es deseable que salga la información ya existente y a la inversa, si se hace una pregunta para acceder a la información ya existente se desea que aparezca la información que se va a introducir. En caso negativo se debería crear un nuevo texto con la nueva información.

2. Para añadir cierta información debemos dar un nombre al texto que vamos a introducir, el del concepto que describe, y luego un texto que describa dicho concepto. Para crear el texto ver las recomendaciones a la hora de crear textos.

3. Los textos de ayuda se almacenan en el archivo Game.txt situado dentro de la carpeta Javy2 en el directorio "/Exes/scripts/config-GameTextual/" y tienen un formato determinado. Todos los textos deben estar antes de la marca "*END*", pues no se leerá nada posterior a dicha marca, y deben tener el siguiente formato:

   ```
   <name>
   Nombre del concepto
   </name>
   <Description>
   Descripción del concepto
   </Description>
   ```

4. Al escribir los textos no se debe sobrepasar el ancho de la pantalla pues Javy2 descarta los caracteres que no entren en la línea del terminal. Además el texto se leerá más claramente en diferentes líneas que en una sola.

5. Una vez introducido el texto con el formato adecuado se puede ejecutar Javy2 (sin recompilar) y realizar las preguntas adecuadas para comprobar que se accede como deseábamos a la nueva información introducida. Si al ejecutar Javy2 se produce un error probablemente no se ha respetado el formato de los textos (los errores propios de la ayuda aparecerán en el fichero "/Exes/scripts/log.txt").

2.7. Recomendaciones en la creación de textos

De nuestra experiencia en la creación de textos hemos extraído una serie de recomendaciones[17] para la creación de nuevos textos, que procedemos a detallar a continuación:

- Los textos no deben ser muy extensos. En textos muy extensos se "difumina" la importancia de las palabras de la pregunta.

[17] Recomendaciones válidas sólo para la ayuda sintáctica

- Hay que tener en cuenta que los textos se deben concebir como respuesta a una pregunta.
- Es importante incluir el nombre del concepto en el texto. Por ejemplo, si estuviésemos ante el concepto Elefante sería bueno comenzar el texto de la forma "Un elefante es..."
- Se deberían omitir los pronombres cuando sea posible. Por ejemplo en la frase "El usuario debe completar el ejercicio propuesto. **Él** debe cazar los bichos...." se debería sustituir "Él" por el nombre al que se refiere ("El usuario")o por algún sinónimo.
- Es recomendable el uso de sinónimos en el texto, pues así se abarcará un mayor número de preguntas. Por ejemplo en el texto "El usuario dispone de un arma. El arma no gasta munición" se podría poner "El usuario dispone de un arma. El rifle no gasta munición".
- Omitir si es posible el uso de sujetos implícitos. Por ejemplo en la frase "El usuario debe completar el ejercicio propuesto. Debe cazar..." sería recomendable poner "El usuario debe completar el ejercicio propuesto. El usuario debe cazar...".
- Realizar pruebas con varias preguntas para acceder a la misma información, y retocar el texto en función de las preguntas cuanto haga falta. Tener en cuenta las respuestas obtenidas a la hora de retocar el texto.
- Tener en cuenta que es prácticamente imposible que el sistema funcione bien para todas las preguntas posibles.
- Tener en cuenta el uso del "more like this". Si realizamos una pregunta en busca de una información concreta y aparece otra usar el" more like this" para ver si la información buscada es la segunda, tercera, cuarta, etc. más similar.

2.8. Mantenimiento

La principal razón de mantenimiento será la actualización de textos detallada en la sección 2.6. No obstante pueden existir razones para cambiar aspectos más importantes del sistema que puedan hacer que se necesite una modificación en nuestro proyecto, como por ejemplo la estructura de directorios.

El sistema usa una serie de archivos para cargar la configuración, los conectores,la estructura de casos y los textos. Estos archivos tienen una ruta fija desde el directorio de trabajo [18], por lo tanto la estructura de archivos debe mantener estas rutas. Si esto no fuese posible en la sección A.2 del apéndice se listan las rutas necesarias y las clases en las que están definidas para facilitar su cambio, aunque esto implica un cambio en el código del sistema y una correspondiente nueva compilación se muestra aquí debido al posible interés para el mantenimiento del sistema.

2.9. Trabajo futuro

Existen diversas ampliaciones que se podrían realizar al sistema, he aquí algunas de las posibles ampliaciones que se podrían realizar:

- **Nuevos idiomas:** Actualmente el sistema de ayuda de Javy2 solo proporciona respuestas a preguntas efectuadas en inglés, si se efectúan preguntas en otro idioma la respuesta obtenida será cualquiera (si reconoce alguna palabra de ese idioma como palabra inglesa) o más probablemente devolverá *"I don't understand you..."*. Para añadir soporte para un nuevo idioma habrá que crear una nueva implementación de los módulos existentes (ver siguiente punto) que analice y busque preguntas en dicho idioma, para ello habría, para empezar, que traducir los textos de la base de casos a este idioma. Posteriormente y una vez creado la nueva implementación habría que añadir la lengua y los mensajes de información en dicho idioma a la clase Messages. Por último habría que modificar los métodos *cambio_idioma(String language)* y *init_modules(String language)* de la clase Help para que cargasen los nuevos módulos si el idioma es el definido.
- **Nuevas implementaciones de los módulos existentes:** Para crear una nueva implementación de un módulo, ya sea para añadir un nuevo idioma o un nuevo método de búsqueda, como por ejemplo algún método estadístico de similitud, hay que crear una clase que implemente el interfaz definido para dicho módulo, y establecer que implementación queremos cargar en el método *init_modules(String language)* de la clase Help.
- **Nuevos módulos:** La creación de nuevos módulos, por ejemplo si queremos que el sistema proporcione ayuda también sobre

[18] El directorio desde el que se llama a la clase Help

conceptos de la programación orientada a objetos, es similar a la creación de una nueva implementación de los módulos existentes. En este caso, sin embargo, habríamos de definir un nuevo interfaz semejante a los ya definidos para los módulos existentes e implementarlo. También habríamos de crear los textos de ayuda para este nuevo módulo. Una vez creado el nuevo módulo habría que añadirlo a la clase Help y modificar su método *init_modules(String language)* para que cargase además de los módulos existentes el nuevo. Además deberíamos modificar el método *String help(String query)* para que preguntase a este nuevo módulo y se quedase con la mejor respuesta de todos los módulos existentes.

- **Ayuda proactiva:** Una posibilidad de ampliación es que el sistema proporcionase ayuda proactiva al usuario, esto es, que el sistema no esperase a que el usuario le hiciese una pregunta por el terminal para proporcionarle ayuda. Podría controlarse la evolución del usuario en la resolución del ejercicio y en caso de que errase se generase una pregunta al sistema para ese error específico. También se podría controlar el tiempo que tarda el usuario en resolver un paso del ejercicio y en caso de ser este excesivo (el usuario estaría atascado) también se generaría una pregunta al sistema específica para el problema que se supone que esta experimentando el usuario. No obstante este sistema es bastante difícil de implementar y requeriría un gran trabajo.

2.10. Conclusiones

En este módulo se ha estudiado e implementado una forma de proporcionar ayuda a un usuario de un sistema de enseñanza interactivo. Se ha realizado un sistema de ayuda sobre la JVM y sobre el juego Javy2 intuitivo y ágil, completamente integrado en Javy2, que responde de manera adecuada a la mayoría de las preguntas que le puedan surgir a un alumno que esté aprendiendo la compilación de Java en la JVM.

Hemos visto la necesidad y la importancia de la existencia de un sistema de ayuda en un sistema de enseñanza interactivo, cumpliendo este módulo con la exigencia requerida, teniendo además una estructura que facilita la ampliación o modificación del sistema, obteniéndose finalmente un módulo completo y versátil que cumple perfectamente los requisitos establecidos por Javy2.

El sistema originariamente partió de un módulo de un proyecto de SSII del curso 2005-2006, ampliándolo y modificándolo hasta que se cumplieron los requisitos establecidos.

Para realizar este módulo ha sido necesario el estudio de diversas herramientas como jColibri, ScriptSystem, o Javy2. Mediante estas herramientas se ha implementado un sistema CBR textual que proporciona ayuda al usuario de Javy2 en lenguaje natural. Este módulo se comparará con el módulo de ayuda semántica para ver cuál de los dos es el más adecuado a las necesidades de Javy2. El estudio comparativo entre ambos módulos se expondrá en el capítulo 4 resolviéndose cuál de los dos módulos se integra en Javy2.

Capítulo 3

Ayuda Semántica

3.1. Objetivo y motivación

El objetivo de este módulo consiste en proporcionar ayuda al usuario sobre el sistema de enseñanza interactivo Javy2 y la metáfora que permite enseñar conceptos de compilación usando este videojuego. El usuario introducirá una pregunta en un terminal del juego Javy2 en lenguaje natural y el sistema deberá proporcionarle la ayuda adecuada a su pregunta. Este módulo se comparará con el módulo de ayuda sintáctica, y tras una serie de pruebas, se integrará en el proyecto de Javy2 aquel que proporcione mejores resultados tanto en rendimiento como en eficacia de respuestas. Para integrar el módulo de ayuda, ya sea semántico o sintáctico, se utilizará el interfaz JNI; puesto que Javy2 está programado en C++, y el módulo de ayuda se realizan en Java.

El sistema Javy2 se ha creado con el objetivo de enseñar a los usuarios (o jugadores) del mismo a utilizar la JVM, utilizando una metáfora y un entorno entretenidos. En él, los enemigos a los que se debe disparar son en realidad porta-recursos, que guardan consigo aquellos recursos que el usuario debe utilizar para construir su código bytecode en el terminal. Así mismo, el ascensor es el medio de traslado de un método a otro, o los exteriores de un nivel simbolizan la referencia al pool de constantes del frame. Todo esto el usuario no tiene por qué saberlo al comenzar el juego.

Por ello y para posibles aclaraciones sobre objetivos o controles es necesaria una ayuda para el juego que trate del contexto y la metáfora.

Se podría escribir un manual con todas estas aclaraciones, pero consideramos mucho más útil un sistema de ayuda interactivo, en el que si al usuario se le ocurre alguna duda, sólo necesite acercarse al terminal y formularla para conocer la respuesta. Es para esto para lo que el sistema de ayuda reconoce el lenguaje natural; y no sea necesario un menú de ayuda contextual por el que haya que ir navegando, provocando una pérdida de tiempo.

Pero hasta aquí explicamos el por qué de un sistema de ayuda, pero, ¿por qué semántico?. La ayuda sintáctica se basa en similitud de textos, es decir, de palabras, entre la pregunta y los textos de ayuda. Es muy efectiva en el caso de textos grandes y técnicos. Pero nosotros no teníamos ningún texto, y además no creíamos que fuesen a ser muy técnicos, puesto que se basan en la metáfora del juego (hay muchas maneras de referirse a los caza-recursos, por ejemplo). Por lo tanto nos pareció interesante probar también con un módulo semántico, que se basa en "aproximación" de conceptos. Es decir, se basan en el significado y no en el lexema en sí.

3.2. Introducción

Este módulo lo comenzamos desde el principio, ya que no había trabajo hecho sobre la ayuda referente al manejo y metáfora de Javy2, ni en implementación ni en textos de ayuda; y es una importante aportación a este proyecto.

Es una implementación basada en redes semánticas. Una **red semántica** o **esquema de representación en Red** es una forma de representación de conocimiento lingüístico en que las interrelaciones entre diversos conceptos o elementos semánticos se les da la forma de un grafo.

La idea principal que hay debajo de las redes semánticas es que el significado de un concepto depende del modo en que se encuentre conectado con otros conceptos. En una red semántica, la información se representa como un conjunto de nodos conectados unos con otros mediante un conjunto de arcos etiquetados que representan las relaciones entre los nodos.[B20]

Para ello comenzamos planteando el tipo de información que contendría la ayuda, y su forma de plantearla. Comenzamos con una fase de adquisición de conocimiento sobre Javy2; y una vez adquirido éste, nos centramos en encontrar la herramienta de representación de dicho conocimiento que se adaptara mejor a nuestras necesidades. Una vez encontrada, pasamos a la fase de representación de conocimiento.

Tras ello, una vez realizada y adaptada la representación, comenzamos con la implementación de la red semántica correspondiente al tratamiento de la entrada como texto en lenguaje natural, y la búsqueda computacional sobre la red semántica.

3.3. Adquisición de conocimiento sobre el juego y la metáfora

Para adquirir el conocimiento sobre el juego y la metáfora nos dedicamos en una primera fase a leer, principalmente, la **Propuesta de documento de diseño para J2VM** [B29] , versión 0.4, pero también nos servimos de los siguientes documentos:

- **Controles en Javy2**, por Marco Antonio Gómez Martín.
- **3D Gameplay for the Java Virtual Machine: from Monkey Island to CounterStrike**, por Pedro P. Gómez Martín, Marco Antonio Gómez Martín y Pedro González Calero.
- **Javy2 para dummies**, por Marco Antonio Gómez Martín.
- Apuntes de la asignatura Ingeniería de Sistemas Basados en Conocimiento, impartida por María Belén Díaz Agudo; de la Facultad de Informática de la Universidad Complutense de Madrid.

Gracias a estos documentos, logramos una idea mucho más clara sobre el uso y la metáfora del juego.

3.3.1. Conocimiento adquirido sobre la metáfora

Escenario: el juego transcurrirá en el interior de una estación espacial, que reflejará el estado de la JVM durante la ejecución del código fuente Java.

Nivel: la estación espacial se dividirá en niveles. Cada nivel estará asociado a uno de los *frames* que deban construirse durante la ejecución del *bytecode*. Los niveles serán independientes entre sí y no existirá comunicación "física" entre ellos (puertas, pasillos, ventanas...). Para cambiar de nivel será necesario activar los *ascensores de la sala de control* (ver más abajo). Desde un punto de vista lógico, todos los niveles tendrán la misma estructura: en todos ellos existirá un conjunto de salas que no variará arquitectónicamente de un nivel a otro (aunque sí podrán variar sus colores, los objetos que aparezcan en ella, etc.), al que denominaremos *sala de control*. El resto de salas o localizaciones serán exclusivas del nivel en que se encuentren y recibirán el nombre de *exteriores*.

Sala de control: en ella se encuentran las metáforas de dos componentes de un *frame*: el *array* de variables locales y la pila de operandos. Cada oponente dispondrá en la sala de control de un *terminal*, un *ascensor de subida* y otro de *bajada*, un *almacén* y una *pila* (ver más abajo), que serán de uso privado (su rival no podrá utilizarlos, aunque sí podrá verlos si se encuentra en el mismo nivel). Asimismo, se distinguirán unos de otros porque en su representación visual destacará el color asignado al avatar del oponente al que están asociados.

Terminal: se emplea para escribir instrucciones de *bytecode*. Cuando un jugador active su terminal, empleando la acción *Usar* sobre él, su pantalla se dividirá en dos regiones. La primera mostrará las partes más relevantes de la sala de control a través de una cámara fija o de varias, si una cámara no fuese suficiente para visualizarlas todas adecuadamente. La segunda región se asemejará a una consola de terminal clásica y será el lugar donde el jugador podrá escribir las instrucciones de *bytecode* (emulando también su funcionamiento clásico). Los cambios producidos en la JVM por el *bytecode* así ejecutado se verán a través de las cámaras de la primera región.

Ascensores: cuando un personaje ejecute una instrucción de *bytecode*

que suponga la creación de un *frame* (p.e.: invokevirtual), se activará su *ascensor de subida* y podrá atravesarlo para llegar a un nuevo nivel. Cuando se ejecute una instrucción que suponga la finalización del *frame* actual (p.e.: return), se activará su *ascensor de bajada*, que le llevará al nivel del que vino. Un ascensor activado tendrá sobreimpuesta una etiqueta indicando el *frame* al que dirige (nombre del método invocado). Cualquier jugador podrá ver esta etiqueta.

Valor: se corresponde con el valor de un tipo primitivo de la JVM y, por tanto, se puede almacenar en una posición del *array* de variables locales o de la pila de operandos. Los valores se representarán mediante cajas del mismo tamaño con una etiqueta gráfica que identifique su tipo primitivo.

Almacén: se trata de un conjunto de cápsulas espaciales contiguas que representará al vector de variables locales del *frame* con que se corresponde el nivel. Existirá una cápsula por posición del vector que podrá contener un *valor*. Cada cápsula estará etiquetada con un número que indicará su posición. Al utilizar la acción *Usar* con una cápsula no vacía, en el inventario aparecerá una copia del valor que había en esa posición. Si el único espacio para valores del inventario ya estaba ocupado, se mostrará un mensaje que avise del problema. Si se utiliza la acción *Usar con*, relacionando el valor del inventario con una cápsula, el valor desaparecerá del inventario y aparecerá en la cápsula, sobrescribiendo el valor que hubiese en ella. Los resultados anteriores sólo tendrán lugar si la *ejecución implícita de una instrucción* (load en el primer caso y store en el segundo) tuvo éxito.

Pila: simboliza la pila de operandos del *frame*. Los *valores* contenidos en ella formarán una columna, estando unos apoyados sobre otros de acuerdo con el orden en que hayan sido apilados. Si la pila no está vacía, se podrán utilizar las acciones *Usar* y *Usar con* para desapilar y apilar el valor del inventario respectivamente. De nuevo, estas acciones sólo podrán tener lugar si la *ejecución implícita de una instrucción* (store en el primer caso y load en el segundo) tuvo éxito.

Botón de reinicio: además de lo expuesto anteriormente, cada oponente tendrá un botón de reinicio privado en la sala de control, que se utilizará para *devolver el hilo de ejecución al último estado correcto*.

Cámara de seguridad: en cada sala de control existirá una cámara de seguridad para cada oponente. Su finalidad será proteger los recursos de un jugador por un tiempo determinado cuando éste cambie de nivel.

Exteriores: los exteriores de un nivel simbolizan la referencia al *pool* de constantes del *frame*. Cuando una instrucción de *bytecode* necesite un valor contenido en el pool (p.e.: métodos, campos, constantes numéricas, etc.), será necesario buscar *recursos* por los exteriores.

Threads independientes: cada oponente ejecutará instrucciones de *bytecode* en su propio hilo de ejecución. Es decir, las instrucciones que ejecuta un competidor sólo afectan a su almacén, pila y ascensores privados. La condición de victoria de una fase es completar la ejecución del *bytecode* antes que el oponente. Ambos oponentes tienen que ejecutar el mismo código intermedio.

Categorías: las instrucciones de *bytecode* disponibles para ejecutar estarán agrupadas en categorías.

Mentores: cada categoría tendrá asociado un NPC distinto, que será capaz de ejecutar todas sus instrucciones (ver más abajo la mecánica de *ejecución de instrucciones*). Los mentores se encontrarán en puntos fijos de la sala de control, lo más cerca posible de los elementos de la misma sobre los que tienen influencia (p.e.: un mentor que pueda ejecutar la instrucción invokevirtual deberá estar cerca de los ascensores).

Agente pedagógico: existirá un NPC especial, Javy, que también se encontrará en una posición fija de la sala de control. La función de Javy es puramente didáctica: orientará al jugador sobre los pasos a realizar para ejecutar la siguiente instrucción de *bytecode*.

3.3.2. Conocimiento adquirido sobre los controles

En este apartado se explicarán las diferentes opciones de controles con los que se espera contar en Javy2. En la actualidad se utiliza sólo el control como en Javy1, aún así, hemos consderado interesante incluir aquí todas las opciones por si en un futuro se pudiesen utilizar todas.

Esta información ha sido adquirida del documento **Controles en Javy2**, por Marco Antonio Gómez Martín.

Cámara y control del usuario

En Javy2 existen hasta tres modos de cámara distintos, aunque es posible que, dependiendo de cómo se haya compilado, sólo pueda utilizarse uno.

Control como en Javy1

En este modo, el personaje y la cámara se controlan como en Javy1. La cámara sigue al personaje, que está en tercera persona, intentando mantenerse cerca de él. Cuando el personaje gira, la cámara no se mueve; cuando el personaje se acerca a la cámara tampoco. Los controles son:

Tecla	**Significado**
w / s	Anda / Retrocede el jugador
a / d	Giro izquierda / derecha del jugador
Botón izdo ratón	Cazar
Flecha izda / dcha	Giro de la cámara

Control como en Mario 64

Este modo de control fue implementado por los becarios del curso 2005/06. La cámara se mantiene siempre detrás del personaje, por lo que cuando éste gira, la cámara lo hace con él. Es imposible ver la cara del jugador. Además, la cámara responde.[a] colisiones con paredes: cuando en el giro choca con una pared, no la atraviesa, pero eleva su altura para que la distancia con el personaje no decrezca demasiado.

Los controles son:

Tecla	Significado
w / s	Anda / Retrocede el jugador
a / d	Strafe izquierda / derecha del jugador
Desplazamiento ratón	Giro del personaje (y cámara)
Botón izdo ratón	Cazar

Cámara libre

Existe una tercera cámara, la cámara libre, que no está ligada al avatar del jugador, sino que debe ser controlada por el usuario. De esta forma, se puede visitar todo el mapa sin tener que mover al personaje.

Este modo de cámara no influye en el control del personaje, por lo que si se activa, el personaje se seguirá controlando como se hacía hasta ese momento (si se viene de cámara como en Javy1, con el teclado, y si se viene de la cámara estilo Mario 64, usando tanto teclas como ratón).

En este modo, las teclas para controlar la cámara son las teclas del cursor para los desplazamientos laterales y avance y retroceso, y las teclas rz "f"para los desplazamientos verticales.

Cambio de cámara

En Javy2 de momento no existen transiciones de cámara automáticas (al cambiar de mapa, etc.), por lo que, si no se indica lo contrario, siempre permanecerá la cámara por defecto. Sin embargo, es posible que la versión se haya compilado permitiendo, utilizando teclas, el paso entre una cámara y otra[1]:

Tecla	Significado
Ctrl-F1	Control como en Javy1
Ctrl-F2	Control como en Mario 64
Ctrl-F3	Cámara libre

[1] Al activarse la cámara libre, su posición y dirección inicial es la actual, por lo que no ocurrirá ningún cambio aparente.

Líneas de depuración

En modo Debug, hemos añadido la posibilidad de ver algunas líneas útiles para ver el comportamiento de ciertas secciones del juego. La siguiente tabla muestra las teclas con las que se activa la visibilidad de esas líneas, y su significado. Que luego existan líneas de esa categoría o no dependerá del mapa.

Tecla	Significado
F1	Grafo usado para búsqueda de caminos
F2	Ruta de cada avatar
F3	Grafo de las manadas
F4	Líneas de las manadas
F12	Líneas genéricas

Inventario y Log

El HUD de Javy por defecto no se muestra. Para mostrarlo, se utilizan distintas teclas que "activanz "desactivan"distintas partes:

Tecla	Significado
I	Inventario
L	Log

Consola de Nebula

Si está creada, se activa y desactiva con la tecla ESC. Es un método válido para salirse de la aplicación sin tener que usar el ratón: en la consola, se teclea "exit", y listo. (También vale con ALT-F4 ;) (hubo que cambiar Nebula 2 para que funcionara).

Cuando la consola está activa, siguen funcionando las teclas con los significados anteriores. Es decir, si se pone "sel /usr", las "s" harán que el personaje se mueva, la "l" que salga el log, y la "r" que se mueva la cámara si estamos en cámara libre.

Interacción con el entorno: usando objetos

En Javy2 se ha incluido la posibilidad de utilizar el objeto que hay al lado del jugador, utilizando la tecla "Usar"("U"), de la misma forma que en Half-Life.

Ejecución en Java

La ejecución de instrucciones de la máquina virtual en Javy2 ya no pasa por interactuar con el entorno como en Javy1, apilando por ti mismo los objetos, etc., sino que se hace tecleando directamente las instrucciones por ti mismo. Para eso, según el documento de diseño [B29], hay que *utilizar* el terminal que hay en el mapa. Eso significa acercarte a él, y pulsar la tecla Usar ("U"). Eso abre un GUI (en 2D), donde se pueden teclear las instrucciones. Para salir de ese GUI, se pulsa la tecla escape.

3.4. Adquisición de conocimiento sobre las herramientas

Para poder realizar este módulo semántico antes estudiamos aquellas herramientas que creímos que nos serían útiles. Entre ellas se encuentra jColibri, que no incluimos en este apartado por hallarse ya explicado en el apartado 2.4.2. En los siguientes apartados estudiamos brevemente el Ontobridge, Protégé, Smore, Aktive y Dublin Core.

3.4.1. Conocimiento adquirido sobre ontologías

La *Ontología* es el estudio filosófico de lo que existe. Desde el punto de vista filosófico, ontología significa "estado del ser", lo que quiere decir, tratado de todo lo que existe. El término ontología en informática hace referencia al intento de formular un exhaustivo y riguroso esquema conceptual dentro de un dominio dado, con la finalidad de facilitar la comunicación y la compartición de la información entre diferentes sistemas. En el contexto de la IA, la ontología se ocupa de las categorías que podemos cuantificar adecuadamente y de la manera en que se relacionan estas categorías con las demás.[B4, B5, B24]

Existen numerosas definiciones de ontologías, entre las que cabe destacar[B25]:

- "Una ontología es un vocabulario acerca de un dominio: términos + relaciones + reglas de combinación para extender el vocabulario". Neches, 1991.
- "Una ontología es la especificación de una conceptualización". Gruber, 1993. (Aquí el término conceptualización se refiere a un modelo conceptual).
- "Una ontología es una especificación formal de una conceptualización compartida". Borst, 1997. (Aquí el término forma se refiere a que es procesable por ordenador).
- "Una ontología es una base de datos que describe los conceptos generales o sobre un dominio, algunas de sus propiedades y cómo los conceptos se relacionan unos con otros". Weingand, 1997.
- "Una ontología necesariamente incluirá un vocabulario de términos y una especificación de su significado (definiciones e interrelaciones entre conceptos) que impone estructura al dominio y restringe las posibles interpretaciones." Uschold-Jasper.

En los siguientes apartados trataremos distintas herramientas para crear y tratar ontologías.

Protégé

Protégé es una herramienta para el desarrollo de Ontologías y Sistemas basados en el conocimiento creada en la Universidad de Stanford. Las aplicaciones desarrolladas con Protégé son empleadas en resolución de problemas y toma de decisiones en dominios particulares. La herramienta Protégé emplea una interfaz de usuario que facilita la creación de una estructura de frames con clases, slots e instancias de una forma integrada.

Las ontologías de Protégé pueden ser exportadas a una variedad de formatos incluído RDF(S), OWL, y a un esquema XML. Protégé está basado en Java, es extensible y proporciona un entorno que flexibiliza un rápido prototipado y desarrollo de aplicaciones. [B21]

La elección de esta herramienta vino de la mano de la recomendación de nuestra tutora del proyecto, con experiencia en el manejo de ontologías, así que nos dispusimos a trabajar sobre la aplicación para crear desde cero la ontología sobre la metáfora de Javy2.

El primer paso fue descargarnos la versión 3.2 beta de la página web [B21], esta versión, a pesar de ser una beta y poder darnos algún problema de inestabilidad, nos pareció la más adecuada para el desarrollo al que nos íbamos a dedicar. Una vez instalado el programa pasamos a una pequeña fase de estudio de las distintas utilidades de la aplicación, para ello hicimos uso del tutorial adjunto a la aplicación así como de los apuntes de la asignatura "Ingeniería de Sistemas Basados en Conocimiento", muy relacionada con todo el mundo de las ontologías. Una vez adquirido el conocimiento necesario para el uso de la herramienta pasamos a crear la ontología y una vez creada utilizamos Protégé continuamente para realizar las distintas evoluciones de la misma.

Sobre la aplicación habría que destacar que se trata de una interfaz sencilla para el manejo y creación de ontologías. Dispone de distintas pestañas donde controlar todos aspectos concernientes a una ontología. Pasamos a describir un poco en detalle todas ellas.

La pestaña principal donde se lanza la aplicación se llama *Metadata*. Al abrir un nuevo proyecto, o uno ya existente, es la ventana donde se puede manejar todo lo concerniente a de qué ontologías (ya existentes) va la nuestra a heredar utilidades. Así como es el lugar desde el cual se controla el namespace asignado a nuestra ontología (ver figura 3.1).

La siguiente pestaña llamada *OWLClasses* es la más importante para el proceso de creación de la ontología, a medida que se van añadiendo conceptos va mostrando el árbol jerárquico que se forma, e internamente genera el archivo OWL. También muestra las propiedades que han sido asignadas a cada concepto, así como las relaciones tanto de herencia, como las condiciones asertadas para que una instancia pertenezca a un determinado concepto. Ésta es la pestaña que más se ha usado a lo largo de la creación inicial de la ontología (vista en figura 3.2).

La pestaña de *Properties* es el lugar donde crear y relacionar las propiedades existentes de la ontología, se pueden crear propiedades de

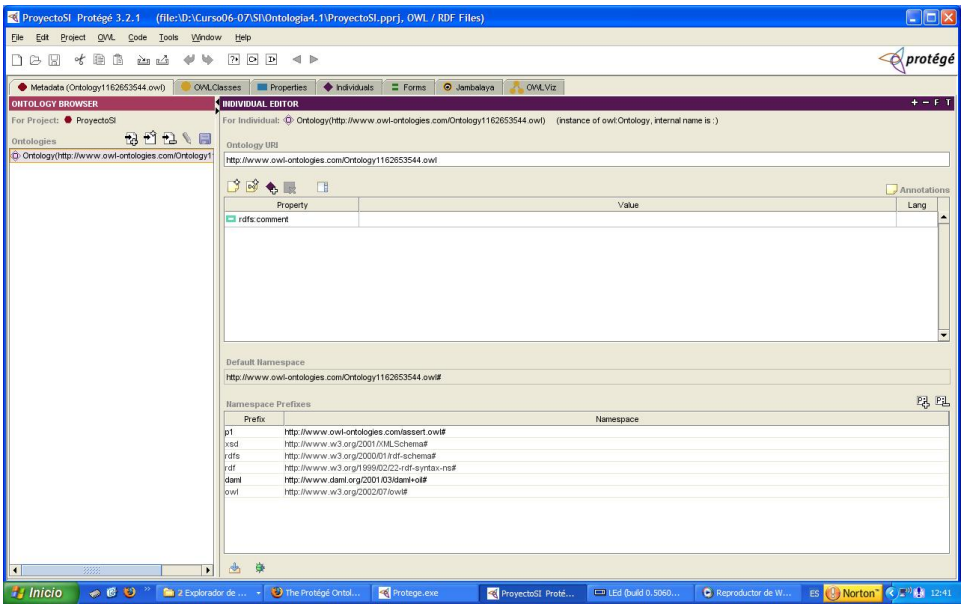

Figura 3.1: Captura de pantalla de la pestaña metadata

objeto de datos o de anotación, como se puede ver en la figura 3.3.

La pestaña *Individuals* es donde se manejan las instancias de los distintos conceptos, también es donde mediante Pellet se puede comprobar la consistencia y taxonomía de nuestra ontología (figura 3.4).

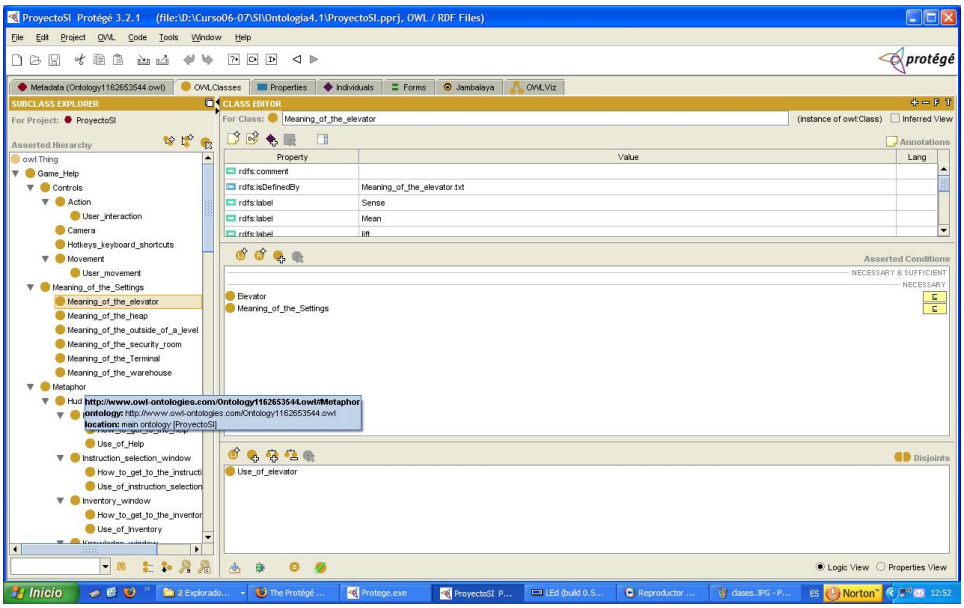

Figura 3.2: Captura de pantalla de la pestaña OWLClasses

Las pestañas restantes son, *Forms* para todo lo que tenga que ver con las propiedades internas de los conceptos, y las dos últimas son dos visores para la ontología. *Jambalaya* (figura 3.5) muestra la ontología como cajas que se encuentran unas dentro de otras siguiendo la relación que tengan entre ellas. Por su parte el *OwlViz* muestra a la ontología de distintas formas. Esta última utilidad no ha servido de mucho ya que la inestabilidad de la beta venía por aquí, y esta pestaña nunca llegó a funcionar como debiera, mal menor por otra parte, al tratarse únicamente de un visor.

Otras herramientas

En este apartado comentamos aquellas herramientas que nos parecieron interesantes, pero que por algún motivo decidimos no usarlas.

Smore

SMORE incorpora los principios de diseño subyacentes indicados por el papel de proporcionar una integración sin costuras de creación de contenidos y anotación. Facilita el marcaje semántico de varios tipos de

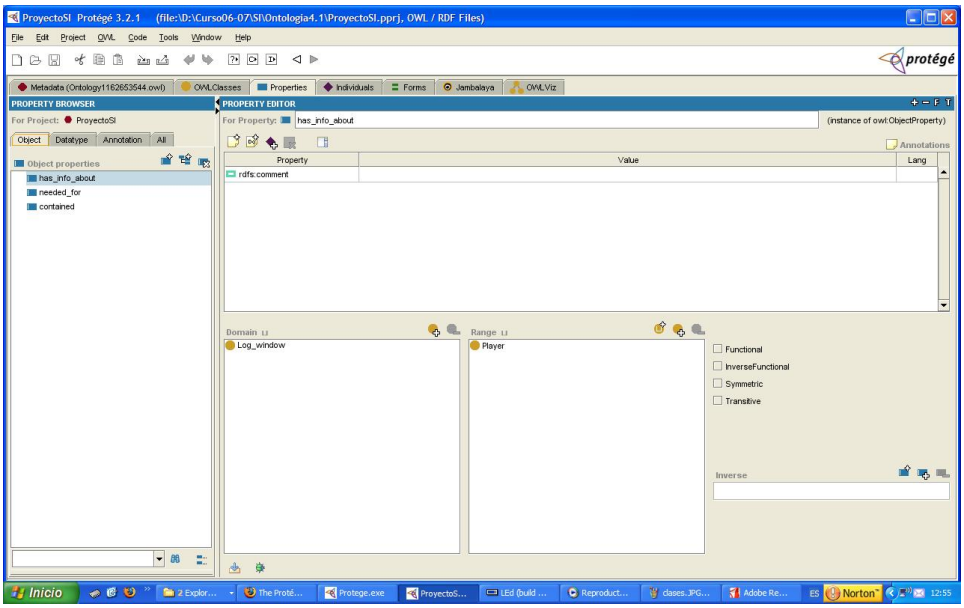

Figura 3.3: Captura de pantalla de la pestaña Properties

medios (fotos, HTML, el correo electrónico) y así proporciona un alto grado de flexibilidad en el uso, la modificación y la extensión de ontologías, todo lo cual puede ser hecho ad hoc.Integra una abundancia de características en un paquete de software, e introduce muchos nuevos rasgos que no están disponibles en otra parte como el portal semántico virtual, que ayuda a los usuarios en el hallazgo de datos relacionados. Así, SMORE es un instrumento fácil de usar y sumamente útil para el marcado.[B21]

La herramienta semántica Smore no nos interesó porque está demasiado dedicado a aplicaciones web, y por tanto no se adaptaba a nuestras necesidades.

Aktive

La herramienta semántica Aktive nos interesó más que Smore puesto que consiste en una herramienta para asociar automáticamente palabras o textos a un concepto. Sin embargo, dado que en nuestro caso los textos iban a ser creados a partir de los conceptos y no al revés, no nos resultaba útil utilizar esta aplicación; que se ha construido para

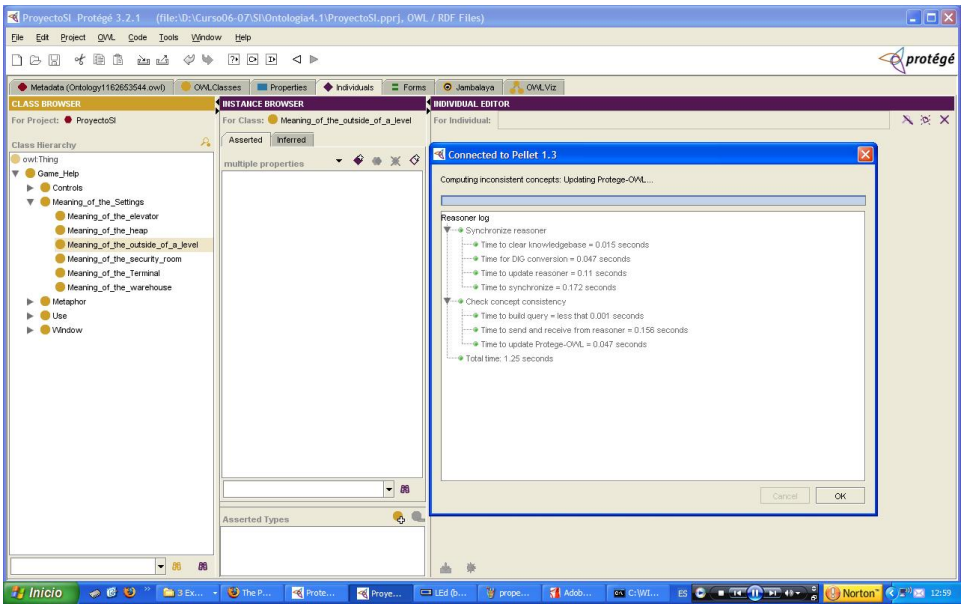

Figura 3.4: Captura de pantalla de la pestaña Individuals

casos en los que los conceptos de una ontología se crean a partir de un texto, como un manual, etc. (Este es el caso de la ayuda sobre la Máquina Virtual de Java).

Dublin Core

La Iniciativa de Metadatos de Dublin Core (DCMI), llamada también **Dublin Core**[B30], es una organización dedicada a fomentar la adopción extensa de los estándares interoperables de los metadatos y a promover el desarrollo de los vocabularios especializados de metadatos para describir recursos para permitir sistemas más inteligentes del descubrimiento del recurso. Es un sistema de **15 definiciones semánticas descriptivas** que pretenden transmitir un significado semántico a las mismas.

Cualquier persona puede utilizar los metadatos de Dublin Core para describir los recursos de un sistema de información. Las páginas Web son uno de los tipos más comunes de recursos que utilizan las descripciones de Dublin Core.[B5]

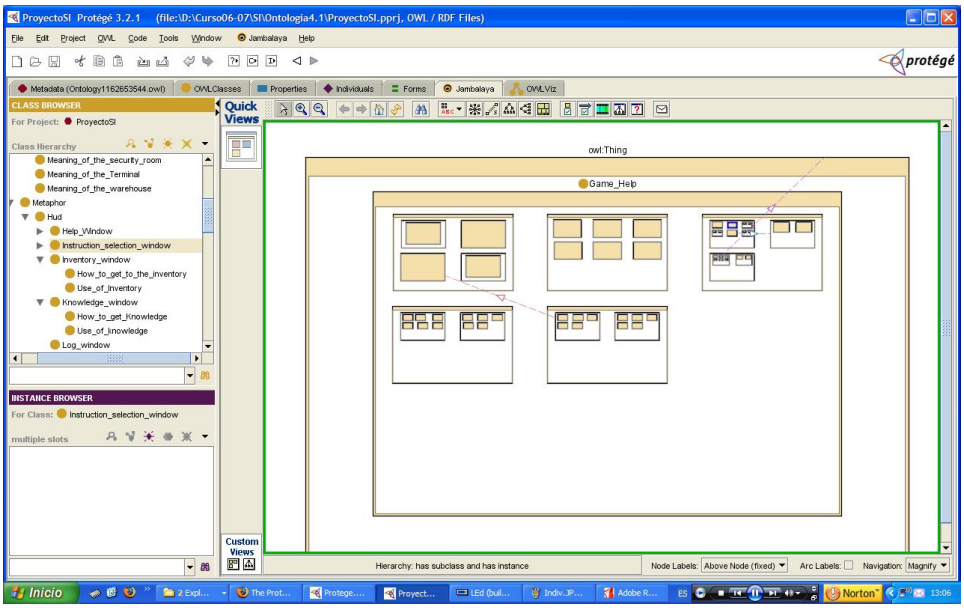

Figura 3.5: Captura de pantalla de la pestaña Jambalaya

La subontología Dublin Core nos permitía añadir una serie de propiedades interesantes, y al estar comúnmente reconocida podría hacer nuestra ontología más portable. Sin embargo, debido al alcance limitado de la información, reducido a Javy2, y a nuestras pocas necesidades con respecto a las propiedades de Dublin Core que podíamos utilizar; creímos que no merecía la pena aumentar la complejidad añadiendo dicha sub-ontología.

3.4.2. Ontobridge

Ontobridge es una librería de Java que facilita el manejo de ontologías. Está basado en Jena 2.4 y es un subproyecto de jColibri. **Jena** es un framework desarrollado por HP Labs para manipular metadatos desde una aplicación Java. Jena permite gestionar todo tipo de ontologías (añadir hechos, borrarlos y editarlos), almacenarlas y realizar consultas con ellas. Jena considera que la abstracción Java del recurso es sólo una vista del mismo.

Características

- Funciona con ontologías locales y online.

- Usa Pellet[B22] (internamente) y cualquier otro razonador DIG (sobre http)[B23].
- Permite la carga de ontologías importadas
- Incluye métodos para acceder y cargar conceptos, instancias y propiedades.
- Simplifica el uso de los elementos de ontología por significado o identificadores.
- Incluye varios ejemplos que muestran aplicaciones comunes e interfaces gráficas.
- Salva ontologías como ficheros OWL RDF/XML.

Arquitectura

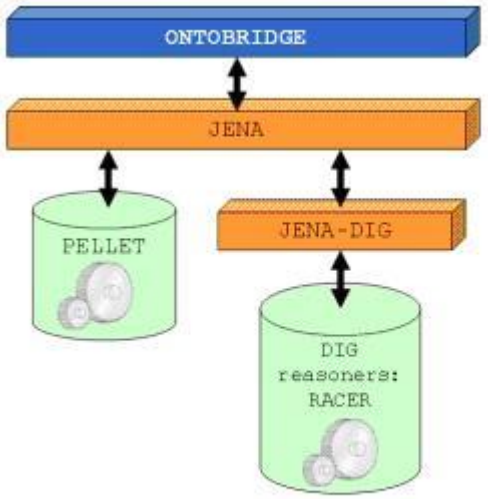

Figura 3.6: Arquitectura del Ontobridge

Ontobridge[B31] es básicamente una máscara para Jena (Figura 3.6).

Nuestro sistema de ayuda se iba a basar en las relaciones entre conceptos, de generales a particulares, y por ello decidimos almacenar dichas relaciones en una ontología.

3.4.3. Wordnet

WordNet [B32] es una base de datos léxica del idioma inglés, organizada semánticamente,cuyo diseño está inspirado en las actuales teorías psicolingüísticas de la memoria léxica humana[2]. Aunque nosotros en último término no lo utizamos, sí nos pareció interesante nombrarlo, por su relevancia en las redes semánticas.

Las palabras están organizadas en una jerarquía de unos 12 niveles, que permite la herencia (basada en las ideas de las redes semánticas).

Considera cuatro categorías gramaticales: verbo, sustantivo, adjetivo y adverbio.

La primera versión (Miller et al., 2004) contenía 95.600 palabras o multipalabras, organizadas en 70.100 significados diferentes o conjuntos de sinónimos, llamados "synsets". Actualmente contiene 152.059 palabras o multipalabras (114.648 sustantivos, 11.306 verbos, 21.436 adjetivos y 4.669 adverbios), organizadas en 115.424 synsets.

Las palabras representadas están relacionadas entre sí por distintas relaciones semánticas, que son:

- sinonimia, antonimia
- "es un": hiponimia, hiperonimia
- "parte de": meronimia, holonimia

3.5. Desarrollo

En el siguiente apartado nos vamos a dedicar a hacer una explicación detallada de lo que ha sido el desarrollo de la Ayuda Semántica. Para ello nos vamos a centrar en los tres factores más importantes de ésta, que son la ontología de conceptos de la metáfora de Javy2, una descripción general del funcionamiento y estructura de la ayuda y por

[2]Propuesta de un Modelo Fundamentado en Análisis de Dependencias y WordNet para RTE WordNet ha sido desarrollada, bajo la dirección de George A. Miller, en el Laboratorio de Ciencia Cognitiva, de la Universidad de Princeton, EE.UU.

último una explicación detallada del funcionamiento de la función de similitud.

3.5.1. Desarrollo de la ontología

Al plantearnos el diseño de la ayuda semántica vimos que la mejor solución para jerarquizar el conocimiento acerca de la metáfora era el de la creación de una ontología que la cubriera en lo máximo posible, para ello mediante el análisis en profundidad del Documento de diseño de Javy2 [B29] extrajimos la metáfora en forma de árbol jerárquico con varios niveles en el que a medida que se profundiza se va cubriendo distintos aspectos de la metáfora.

A continuación pasamos a describir el proceso seguido para la generación de la ontología así como sus distintas evoluciones a o largo de la fase de implementación.

Primera fase

Creamos una primera versión de la ontología en la que no existía herencia múltiple; y nos dimos cuenta de que muchos de nuestros conceptos dependían de distintas ramas y por tanto de distinta herencia, lo que las relacionaba de alguna manera. Por ejemplo, este era el caso de todas las descripciones, las cuales se relacionaban en el tipo de información buscado aunque los objetos a describir no se relacionaran en absoluto entre sí. Por ello creamos una segunda versión de la ontología en la que añadimos los siguientes conceptos que relacionaban el tipo de información:

- Descripción
- Uso
- Cómo Llegar, para los conceptos relacionados con la ventana.

Tras esta segunda aproximación nos dedicamos a depurar algunas relaciones que no nos quedaban del todo claras. Después pasamos a añadirle las propiedades a los conceptos. Íbamos a añadir dos tipos de información:

- Ideas relacionadas.
- Textos de ayuda para el usuario.

Ideas relacionadas:

Para añadir las ideas o conceptos relacionados con cada nodo de la ontología nos decidimos por los **rdfs:label**, por su facilidad de uso y acceso. Antes barajamos la posibilidad de distintas estrategias como la del uso de la Dublin Core Metadata, pero concluimos que el uso de esta utilidad para nuestro propósito era como "matar moscas a cañonazos".

Por tanto mediante los **rdfs:label** añadíamos los conceptos sinónimos a otro existente en la ontología y a todos los hijos, porque tal y como habíamos creado el esquema los textos de ayuda se asociarían sólo a las "hojas". El problema que encontramos con este método es que a la hora de cambiar el idioma de la ayuda habría que traducir cada "label"; pero tras consultar con gente entendida en la materia, aceptamos las consecuencias de ello. Para un dominio más amplio de la información este riesgo debía ser tratado, pero en nuestro caso fue factible asumirlo. En caso de añadirle un nuevo idioma se podría interponer entre la interfaz con el usuario y el módulo semántico un traductor que convirtiera la pregunta a inglés.

Textos de ayuda:

Para añadir los textos estuvimos estudiando si añadirlos dentro de la ontología como el valor de una determinada propiedad o si en lugar de ello asociar los textos a los conceptos mediante URL (URI). El inconveniente de esta última opción sería la posible lentitud del módulo, pero ganaríamos modularidad y facilitaríamos el mantenimiento, ya que cambiar un texto sólo consistiría en cambiarlo en el fichero asociado por la URL, o directamente sustituir dicho archivo por otro con el mismo nombre. Incluso facilitaría el cambio de idioma. Por todo esto a pesar de la posible ralentización decidimos usar la segunda opción. Para ello utilizamos la propiedad **rdfs:isDefinedBy**, que añadimos a cada "hoja" de la ontología, y cuyo valor es la dirección relativa del fichero de texto de la ayuda de cada nodo.

Nuestra ontología tenía los nombres de nodos en español, aunque los valores de las etiquetas y el uso de la ayuda fuese en inglés. No nos pareció muy correcto, así que los traducimos al inglés, en otra de las evoluciones realizadas. Nos encontrábamos ya en la versión 3.0 de esta ontología.

Estructura General:

La estructura general que se ha seguido a lo largo de todo el desarrollo incluyendo diversos cambios es la siguiente.

La raíz principal de la ontología se llama *GameHelp*, este es el concepto padre del resto de la ontología y a su vez es hijo de *owl:Thing*. *GameHelp* tiene cinco hijos en los cuales se intenta cubrir todas las posibles dudas que se le pueda plantear a un usuario de Javy2. Para realizar la elección de estos cinco hijos nos planteamos la situación de estar jugando a Javy2, y cuáles serían nuestras preguntas a medida que fuéramos jugando. Esto lo hicimos usando de apoyo el documento de diseño inicial que se nos proporcionó [B29]. Cada una de estas cinco clases o conceptos (figura 3.7) cubre un campo determinado de preguntas que se le pueden plantear a un usuario. A lo largo de esta sección se irán mostrando distintas imágenes que muestren la estructura de la ontología en cada nodo, una general de toda ella es demasiado amplia para ser incluida en el documento, por ello mostramos en la figura 3.7 solo el primer nivel.

Figura 3.7: Estructura del primer nivel de la ontología

- **Controls**:

  Cubre el conocimiento referido al movimiento del personaje así como todo lo concerniente al uso del teclado, consideramos que era una parte importante que cubría una gran cantidad de preguntas que se podían realizar a cerca de la interacción del usuario con el juego. La estructura de este concepto es la mostrada en la figura 3.8.

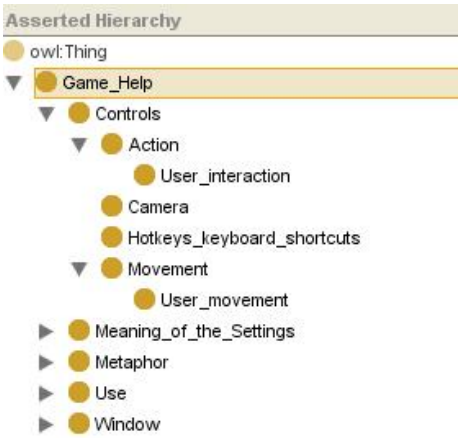

Figura 3.8: Estructura del nodo controles

- **Meaning of the settings**:

  Con este concepto y sus hijos se pretende cubrir todas las posibles preguntas referentes al significado, dentro de la metáfora, de cualquiera de los componentes del escenario. En un principio también existía una clase con las descripciones físicas de las ventanas del Javy2, pero después de modificar varias versiones de la ontología se vio que esta información no era necesaria para el usuario. Así que se eliminaron las descripciones físicas. La estructura es la de la figura 3.9.

- **Metaphor**:

  Este concepto y sus hijos tratan todo lo referente a la metáfora, es el campo más extenso ya que dentro de lo que es la metáfora

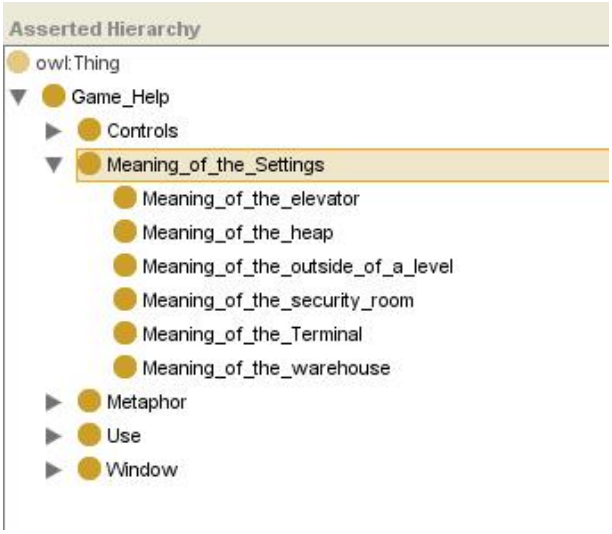

Figura 3.9: Estructura del nodo MeaningOfTheSettings

del juego nos podemos encontrar con distintos enfoques, los que nosotros hemos considerado se muestran en la figura 3.10.

- *HUD:* Cubre todas las ventanas existentes dentro del Javy2. Su estructura se puede consultar en la figura 3.11.
- *Player:* Todo lo que se refiere a la metáfora respecto al jugador lo cubre Player. Esta clase es muy extensa en su interior, tiene dos sub-conceptos principales llamados Npc y UserPlayer:
  - NPC: cubre todo el conocimiento necesario acerca de los personajes ajenos al usuario que intervienen en el desarrollo del juego, estos son los avatares o Enemies que son los que portan los recursos necesarios para el usuario, los mentores (Mentors) que son los que colaboran con el usuario, y el llamado Jugador Virtual (Virtual_Player) que cubre a la opción tanto de competición en red o competición contra un jugador controlado por la aplicación (figura 3.12).
  - User Player: Esta clase se encarga de todo lo referente a las posibilidades del jugador durante el juego. Tiene las subclases, mostradas en la figura 3.13.
- *Setting:* Este sub-concepto sigue perteneciendo a la metáfora y cubre todo el conocimiento referido al escenario del juego

Figura 3.10: Primer nivel del nodo Metáfora

y los distintos lugares en los que se desarrolla la acción. Se diferencia entre escenarios interiores y exteriores. Estos a su vez se verán explicados por los conceptos uso y significado de cada escenario. Como decíamos con anterioridad el significado se refiere al sentido de dicho escenario dentro de la metáfora del juego. (Mostrado en la figura 3.14).

- **Use y Window**:

Estas dos últimas clases (conceptos) cubren todos los conceptos referidos a los usos, tanto de escenarios como de ventanas, y todo el conocimiento referido a las ventanas (Como llegar y uso) agrupando todo el significado que a lo largo de la ontología trata sobre ellas. La razón de agrupar todos estos conceptos que ya han sido vistos con anterioridad en la ontología es para controlar tanto la consistencia como la reutilización de conceptos. Al tener varios conceptos con distintos padres que estaban relacionados, era necesario mantenerlos bajo un mismo padre común para mostrar esta relación. La estructura de estos dos últimos conceptos es mostrada en las imágenes 3.15 y 3.16.

Figura 3.11: Segundo nivel de Metáfora − > Hud

Diccionario de Conceptos Hoja

A continuación realizamos una explicación detallada de los conceptos hoja de la ontología. Explicamos sólo las hojas ya que son los únicos conceptos que tienen textos de descripción asociados. Es decir, son los conceptos donde se guarda la respuesta devuelta al usuario al identificar la pregunta realizada con dicho concepto. Se hace una excepción en los casos en los que llegue con explicar al concepto padre de varios hermanos para entender a estos últimos. Se realiza en orden alfabético para facilitar la búsqueda de los términos que no se comprendan al explorar la ontología o la estructura de la misma en el punto anterior de este documento:

- **Actual level of the virtual player**: Se refiere al nivel de conocimiento que tiene a cerca de los ejercicios propuestos el jugador virtual (o jugador rival cuando exista modo multijugador).
- **Box**: Se refiere a las cajas que el jugador se encuentra desperdigadas por la nave, las cuales contienen recursos de todo tipo, necesarios para la realización de los ejercicios.

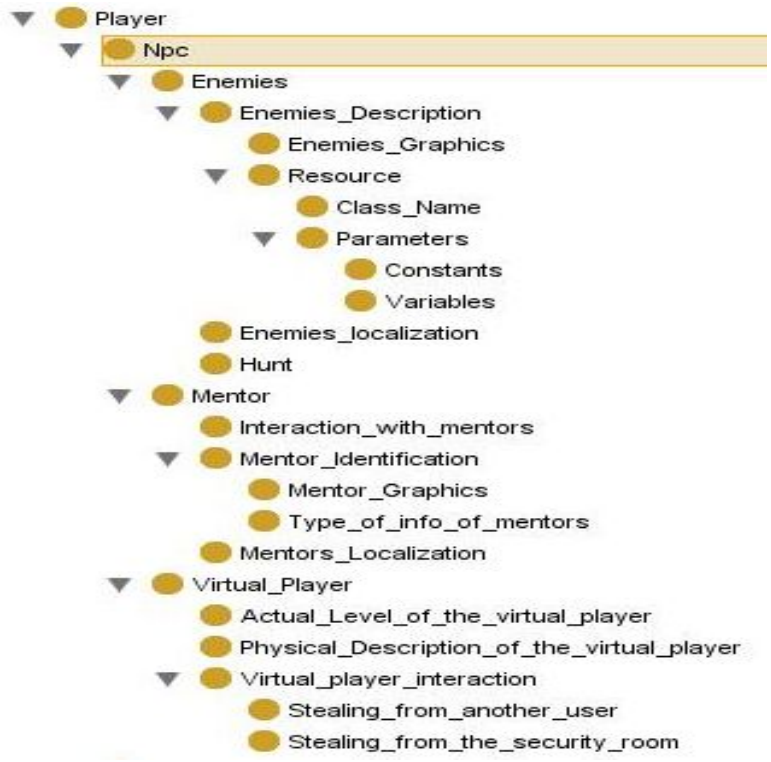

Figura 3.12: Segundo nivel de Metáfora - Player - Npc

- **Camera**: Se refiere al manejo de la cámara o puntos de vista durante el juego.
- **Class Name**: Se refiere al tipo de recurso que puede encontrarse un usuario en la nave, que son nombres de clase que ha de usar para resolver ejercicios.
- **Constants**: Se refiere al tipo de recurso que puede encontrarse un usuario en la nave, que son constantes necesarias para resolver los ejercicios.
- **Enemies Graphics**: Se refiere a la descripción física de los aliens (enemigos,malos..) que el jugador se encuentra por la nave, de utilidad para poder reconocerlos y diferenciarlos de los avatares.
- **Enemies Localization**: Se refiere a la posible localización dentro de la nave donde posiblemente se pueda encontrar a los aliens.
- **Experience**: Explica cómo influye la experiencia concreta de un jugador (nivel de conocimiento) respecto a los ejercicios propuestos.

Figura 3.13: Segundo nivel de Metáfora - Player - UserPlayer

- **Final target of the game**: Explica cuál es el objetivo final del juego.
- **Hotkeys and Keybords Shortcuts**: Cubre toda la información necesaria para comprender el uso del teclado así como de las combinaciones posibles de teclas rápidas para acceder a ciertas funcionalidades del juego.
- **How to get** : Este es uno de los ejemplos de concepto padre que llega para explicar a todos sus hijo, ya que todos ellos se refieren a la manera de llegar o visualizar la ventana a la que pertenece cada concepto hijo (Help, Instruction selection screen, Inventory, Knowledge, Source Code).
- **How to get Knowledge**: Explica cómo llegar a la ventana que representa el nivel de conocimiento actual del jugador (es posible que esto ya no exista en la versión actual de Javy2).
- **Hunt**: Cubre el conocimiento necesario a cerca de cómo se realiza y para qué, la captura mediante el arma de recursos de enemigos.
- **Interaction with mentors**: Explica cómo se interactúa con los mentores.
- **Log window**: Es la única ventana que no tiene ni concepto hijo *uso* ni concepto hijo *como llegar* ya que está siempre presente en la consola del Terminal. El concepto *Log window* explica que sentido tiene dicha ventana.

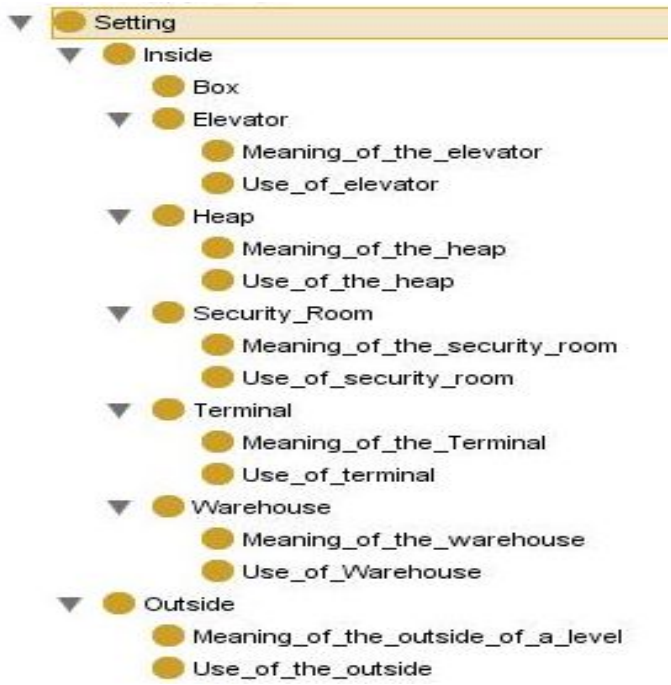

Figura 3.14: Segundo nivel de Metáfora - Setting

- **Meaning of the X** : Con este concepto padre consideramos cubiertos la explicación de todos sus hijos, excepto la de Meaning of the Outside of a level. El concepto explica el significado de la parte del escenario concreta a la que pertenece. Dichas partes son las siguientes: Elevator, Heap, Security room, Terminal, Warehouse, Outside of a level.
- **Meaning of the Outside of a level**: Se refiere a la interpretación que tiene dentro de la metáfora del juego la parte externa de la nave espacial. En concreto se interpreta como el lugar donde se encuentran el "pool" de constantes del frame de la pila.
- **Mentor Graphics**: Describe físicamente a los mentores del juego, para poder diferenciarlos de los aliens.
- **Mentor Localization**: Se refiere a la posible localización dentro de la nave donde posiblemente se pueda encontrar a los mentores.
- **Particular Exercise**: Explica cuál es el objetivo de la realización de ejercicios progresivamente más complicados.
- **Physical Description of the virtual Player**: Describe físicamente al jugador virtual.

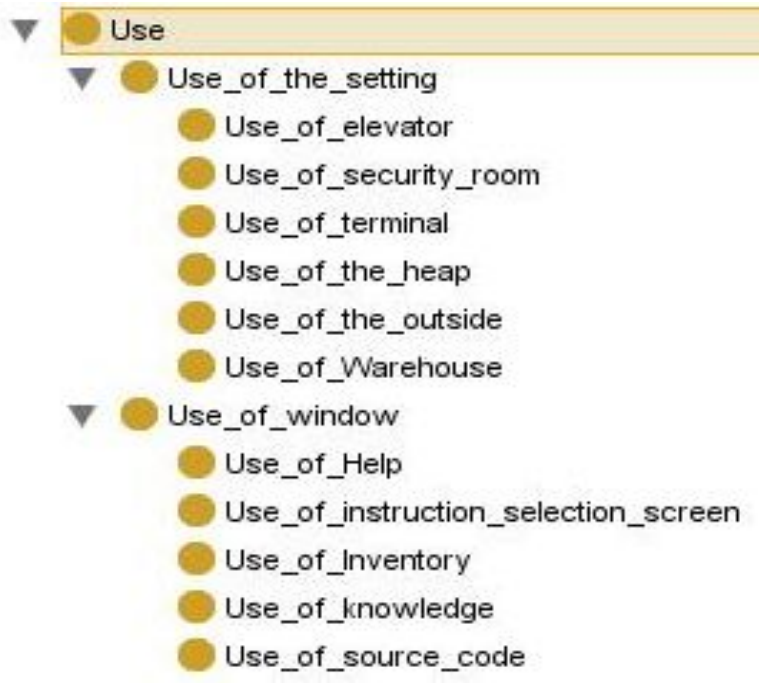

Figura 3.15: Estructura del nodo Use

- **Resource**: En el concepto Recurso encontramos lo que un avatar (Enemie) puede portar, según el documento de diseño [B29], nombres de clase y parámetros. Estos últimos pueden ser constantes o variables.
- **Stealing from another User** : Explica cómo se realiza el robo de recursos a jugadores virtuales.
- **Stealing from the security room**: Explica cómo se realiza el robo de recursos a la cámara de seguridad de los jugadores rivales.
- **Type of info of Mentors**: Explica qué posible información nos puede aportar un mentor para la consecución de un determinado ejercicio.
- **Use of X**: Con Use of nos referimos a, una vez visualizado la ventana, qué función desempeña y cómo manejarla. Cubre la explicación de sus conceptos hijos. Estos son: Help, Instruction selection screen, Inventory, Knowledge, Source Code, Elevator, Heap, Security room, Terminal, Warehouse, Outside of a level. Este último lo explicamos a continuación porque entendemos puede resultar confuso.

Figura 3.16: Estructura del nodo Window

- **Use of the Outside** : Se refiere a qué es lo que el usuario se encuentra al salir al exterior de la "nave espacial", y cómo utilizar estos recursos que se encuentra.
- **Variables**: Explica el uso y el sentido de encontrarse recursos que son variables.
- **Weapon**: Explica en qué consiste y cómo se usa el arma del jugador.

3.5.2. Ayuda semántica en Java

Después de descubrir los detalles del desarrollo de la ontología en este apartado veremos la descripción de la ayuda, así como la estructura general y la función de similitud. Para detalles de implementación, recomendamos consultar el apéndice B.

Descripción general

El módulo recibe como entrada una pregunta en inglés en lenguaje natural. Esta pregunta es desglosada. Elimina los signos, palabras su-

perfluas (What, is, the...), trata los plurales y busca en la ontología los conceptos más apropiados, acordados por una función de similitud (Ver subsección 3.5.3). Estas respuestas son ordenadas de mayor a menor similitud. El sistema devuelve como salida el texto asociado al primer concepto de la lista. Si el usuario pide más información sobre su pregunta (con una serie de frases clave como "more", "more like this", "tell me more"...), el sistema devolverá el siguiente concepto de la lista, y así consecutivamente. Para salir, el usuario debe escribir "exit".

Estructura general

La ayuda semántica se compone de dos módulos o paquetes: **help.game** y **motor**. **help.game** contiene la interfaz **GameGenericHelp**, que hará posible la comunicación con Javy2. **motor** es el paquete que contiene toda la funcionalidad. En el apéndice B se muestra toda la estructura detalladamente.

3.5.3. Función de similitud

La función de similitud del sistema se basa en dos factores:

1. **Hasta qué nivel de ascendencia se repite esa palabra.** Una palabra tendrá una relación más específica con un concepto si dicha palabra sólo se encuentra asociado a ese concepto, y más general cuanto que el padre del concepto en cuestión también la tenga, puesto que eso implica que todos los hermanos tendrán asociada esa misma palabra. Por ello la similitud de una palabra con respecto a los padres será uno dividido entre el número entero correspondiente al número de ascendientes que tienen la palabra asociada, desde la hoja que tenga dicha palabra. Por ejemplo, en nuestra ontología, la palabra *Window* es muy general, mientras que, por ejemplo, *Inventory* es más particular. Véase la figura 3.16, en ella, todos los conceptos hijos del concepto *Window* tendrán asociada la palabra *Window*. Sin embargo, sólo tendrán asociada la palabra *Inventory* los conceptos *How_to_get_to_the_inventory* y *Use_of_Inventory*. No es equitativo, por tanto, asociar a cada palabra el mismo valor. Ante la pregunta "Where is the inventory window?", la función de similitud sobre ascendientes devolverá,

para la palabra “inventory”, un valor mayor que para la palabra ”window”.

2. **El número de palabras de la query que se encuentran como palabras relacionadas del concepto.** Es decir, la similitud de cada concepto hoja de la ontología será la suma de las similitudes (desde el punto de vista de ascendencia, como hemos visto en el punto anterior) de las palabras de la query con dicho concepto.

Ejemplo

Imaginemos que queremos saber cómo conseguir las constantes en Javy2, es decir, a quién tenemos que atacar para consegirlas; le hacemos al terminal la siguiente pregunta:
who is the npc who carry the constants?
Ante esta pregunta, el sistema encuentra similitud con los siguientes conceptos:

- ♯Enemies_localization
- ♯Physical_Description_of_the_virtual_player
- ♯Enemies_Graphics
- ♯Actual_Level_of_the_virtual_player
- ♯Mentors_Localization
- ♯Stealing_from_another_user
- ♯Type_of_info_of_mentors
- ♯Hunt
- ♯Interaction_with_mentors

Todos ellos relacionados con la palabra *NPC* (Ver figura 3.11). Sin embargo, con la palabra *Constant*, que es la que a nosotros nos interesa más, puesto que es más específica, sólo hay un concepto relacionado:

- ♯Constants

Obviamente la información que queremos es la referida al nodo ♯*Constants*. Con la función de similitud sobre ascendientes lo conseguimos. Lo siguiente es la extracción de las similitudes entre conceptos de la frase anterior:

```
Pregunta:

who is the npc who carry the constants?
1.-Similitud entre ascendientes del concepto
        #Enemies_localization
         Número de nodos ascendientes con la
                                palabra npc: 3.0.
                        Similitud: 0.33333334
1.-Similitud entre ascendientes del concepto
#Physical_Description_of_the_virtual_player
         Número de nodos ascendientes con la
                                palabra npc:
                                        3.0.
                        Similitud: 0.33333334
1.-Similitud entre ascendientes del concepto
#Stealing_from_the_security_room
         Número de nodos ascendientes con la
                                palabra npc: 4.0.
                        Similitud: 0.25
1.-Similitud entre ascendientes del concepto
#Enemies_Graphics
         Número de nodos ascendientes con la
                                palabra npc: 4.0.
                        Similitud: 0.25
1.-Similitud entre ascendientes del concepto
#Actual_Level_of_the_virtual_player
         Número de nodos ascendientes con la
                                palabra npc: 3.0.
                        Similitud: 0.33333334
1.-Similitud entre ascendientes del concepto
#Mentors_Localization
         Número de nodos ascendientes con la
                                palabra npc: 3.0.
                        Similitud: 0.33333334
1.-Similitud entre ascendientes del concepto
#Mentor_Graphics
         Número de nodos ascendientes con la
```

```
                                palabra npc: 4.0.
                        Similitud: 0.25
1.-Similitud entre ascendientes del concepto
#Stealing_from_another_user
        Número de nodos ascendientes con la
                                palabra npc: 4.0.
                        Similitud: 0.25
1.-Similitud entre ascendientes del concepto
Type_of_info_of_mentors
        Número de nodos ascendientes con la
                                palabra npc: 4.0.
                        Similitud: 0.25
1.-Similitud entre ascendientes del concepto
#Hunt
        Número de nodos ascendientes con la
                                palabra npc: 3.0.
                        Similitud: 0.33333334
1.-Similitud entre ascendientes del concepto
#Interaction_with_mentors
        Número de nodos ascendientes con la
                                palabra npc: 3.0.
                        Similitud: 0.33333334
1.-Similitud entre ascendientes del concepto
#Constants
        Número de nodos ascendientes con la
                                palabra constant: 1.0.
                        Similitud: 1.0

Concepto en posición 0:
#Constants
        Similitud: 0.025641026


*CONCEPTO: Constants

*DESCRIPCION: A resource carrier can carry five different
types of resources. Each type of resource is identified
by a coloured sphere over the resource carrier.The colour
of the sphere that carries a constant is green.Even that
you know that the animals which have the green ball over
them carry a number you don't know what number is it, so
you may have to hunt more than one to find the number you
```

need .

Tiempo␣tardado␣por␣ayuda␣semantica:␣32␣ms

Como podemos ver, todos los conceptos asociados a la palabra *NPC* tienen una similitud inicial de 0.33333334, puesto que son tres los ascendientes asociados a dicha palabra, mientras que la similitud inicial del concepto asociado a *Constants* (♯*Constants*) es de 1, ya que ninguno de sus antecesores tiene asociada la palabra. Por tanto, el texto más similar es el de *Constants*.

3.6. Ejemplos de ejecución

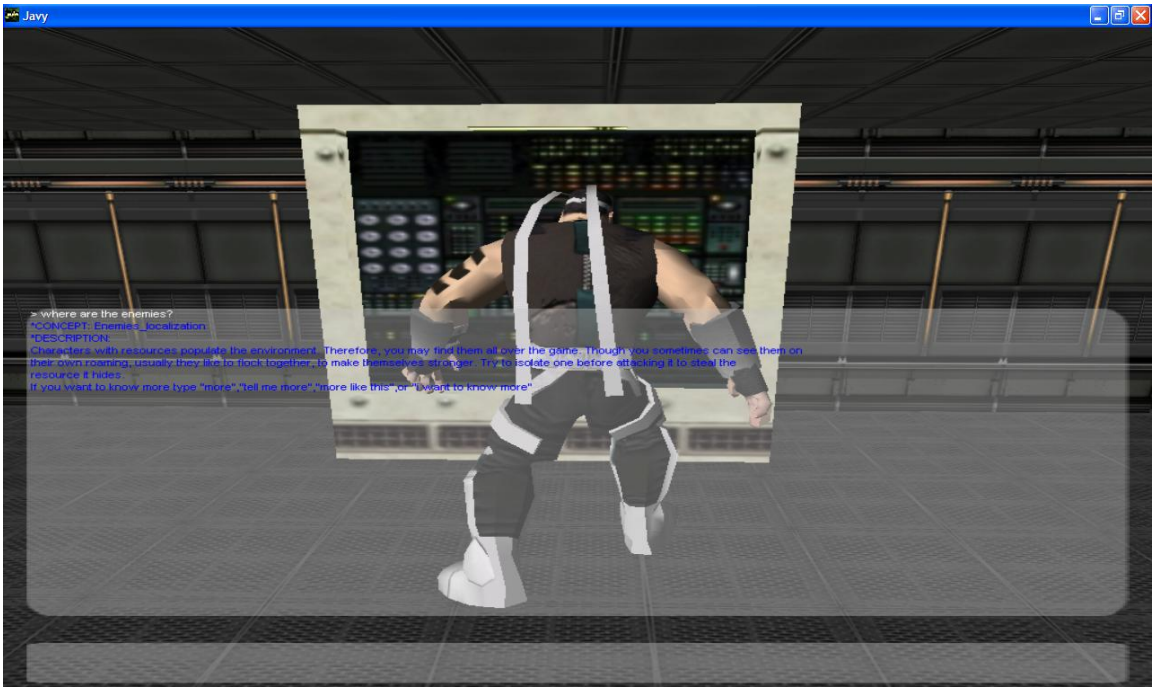

Figura 3.17: Respuesta a la pregunta "Where are the enemies"

Aquí incluimos tres capturas de pantalla del módulo ya integrado en Javy2, con distintas preguntas formuladas:

- **"Where are the enemies?"**: Figura 3.17.
- **"tell me about the keyboard sortcuts"**: Varias preguntas, con ejemplo de la opción *Tell me more*, Figura 3.19.

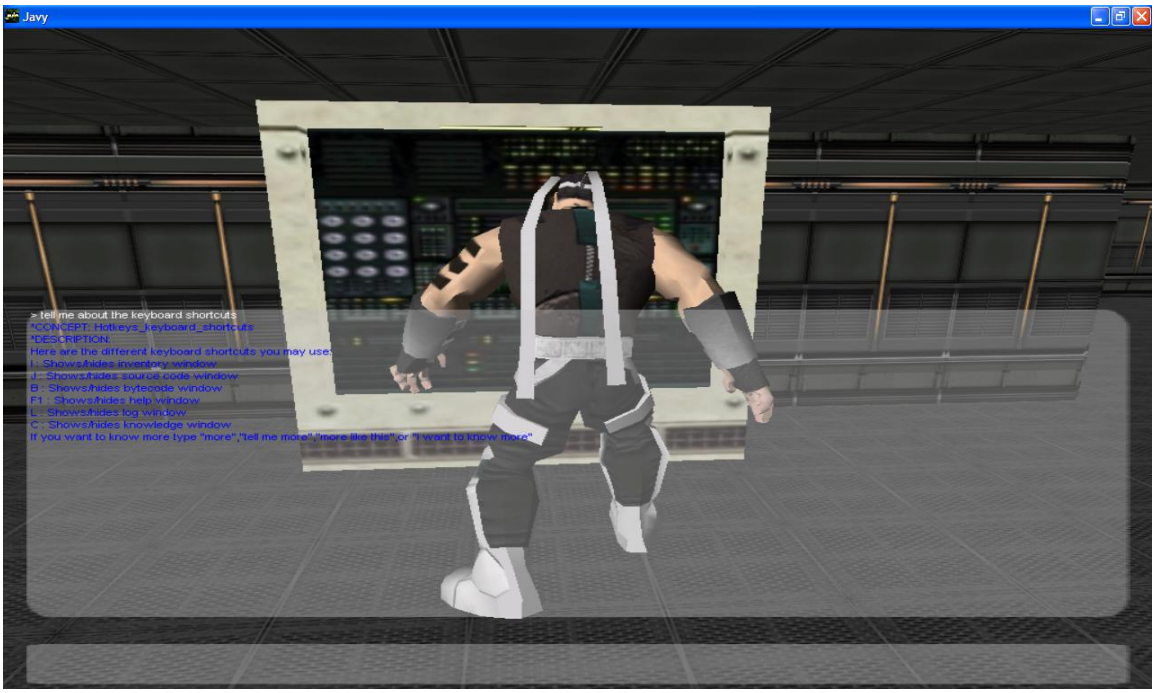

Figura 3.18: Respuesta a la pregunta "tell me about the keyboard sortcuts"

- **"tell me about the keyboard shortcuts"**: Figura 3.18.

3.7. Trabajo Futuro

A lo largo de todo este módulo de la ayuda semántica hemos ido comentando posibles ampliaciones a realizar para ampliar o mejorar el funcionamiento de la ayuda semántica. En este apartado nos disponemos a juntarlas todas y explicarlas con más profundidad. Evidentemente seguro se nos escape alguna, ya que el trabajo en este campo de la búsqueda semántica es muy amplio, pero nosotros describimos, a continuación, las que opinamos son de mayor interés para la aplicación. Antes de comenzar hay que tener en cuenta, que en el caso de las ayudas, estas ampliaciones, al funcionar de manera sintáctica y semántica según para que preguntas, tienen que añadirse a las ya comentadas en la sección 2.9 De hecho varias de las comentadas en dicha sección podrían ser incluidas en esta también, y viceversa.

- Como llevamos comentando a lo largo de todo el documento, Javy2 es una aplicación en constante evolución, y por tanto tam-

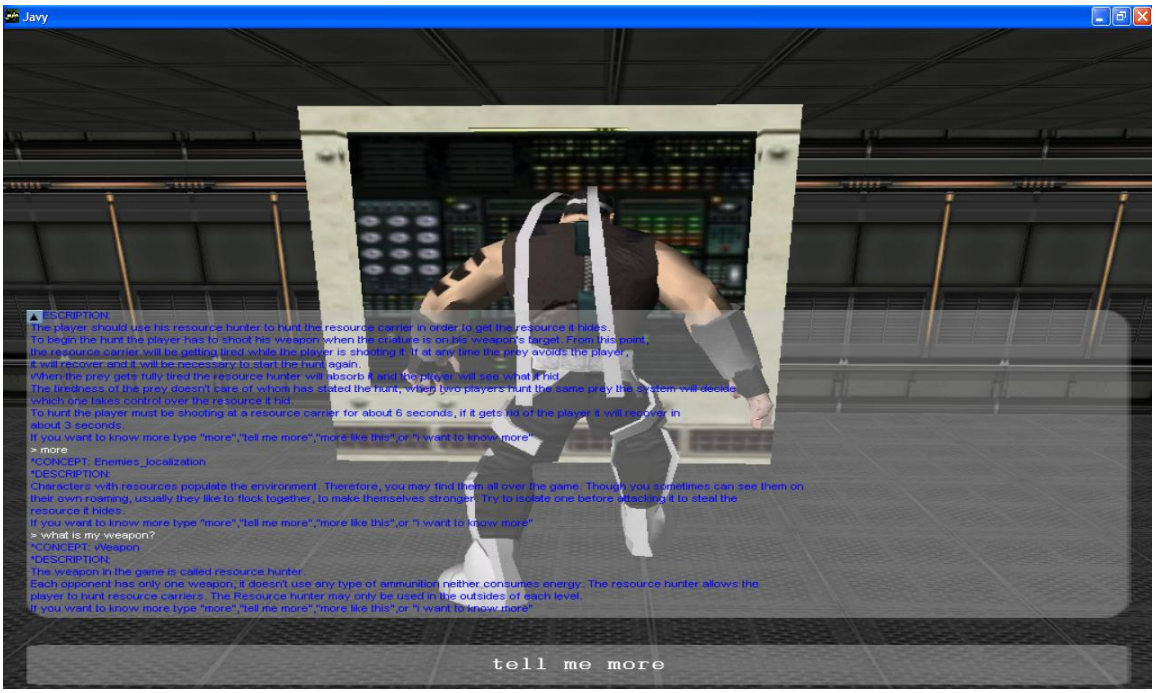

Figura 3.19: Varias preguntas junto con una petición "Tell me more"

bién la metáfora muta con el paso del tiempo y se le añaden o quitan elementos continuamente. Por ello es importante que la manera de incluir estos cambios en el módulo de ayuda este clara y sea lo más sencillo posible. Para incluir estos posibles modificaciones sobre la ayuda habrá que seguir los siguientes pasos:

1. Incluir los nuevos conceptos que cubran esta modificación de la metáfora en la ontología sobre la misma. Para ello analizar detenidamente el sentido del concepto para poder colocarlo en el sitio adecuado dentro de ésta. En caso de que la ontología no cubriera dicho conocimiento añadir también los conceptos padre necesarios para que pase a hacerlo.

2. Una vez cubierto el concepto en la ontología, habrá que crear el texto de ayuda que valga como respuesta para dicho concepto, estos textos se han de confeccionar pensando que será la ayuda mostrada al usuario en caso de que la pregunta se relacione con dicho concepto, por ello debe cubrir lo máximo posible las posibles dudas que un usuario pueda tener a cerca de él. Los textos no han de tener un formato concreto

a diferencia de la Sintáctica, pero en los confeccionados por nosotros hemos seguido el mismo formato en todos para que dicha solución quede lo más elegante posible. Por ultimo, respecto de los textos, habrá que ubicar el texto en la misma carpeta que el resto dentro del Javy2 e incluir el nombre del archivo asociado como un *rdfs:isDefinedBy* al concepto en cuestión.

3. Llega el punto de pensar e indagar lo máximo posible para reunir el mayor número posible de sinónimos del nuevo concepto. Cuantos más se encuentren más cubierta estará la pregunta al respecto. Como ya se explicado en este documento, los sinónimos se añaden el concepto de la ontología como *rdfs:label*.

4. En este punto la ayuda semántica ya estaría preparada para responder a preguntas relacionadas con el nuevo concepto. Comentamos como ejemplo el caso real de incluir la capacidad de usar rayos x por parte del personaje de Javy2, para añadir esta facultad a la ayuda crearíamos un nuevo concepto sobre la ontología, con el Protégé, llamado *XRay* que encuadraríamos bajo el concepto *User Player* o incluso bajo *Weapon* añadiendo uno más llamado Gun que incluyera el conocimiento que hasta ese momento sostenía *Weapon*. Una vez realizado esto solo quedaría crear el texto de ayuda asociado y modificar el de los controles para que incluya como se usa esta nueva capacidad del personaje. Este es un caso real que no está incluido en la ontología porque surgió una vez cerrada la implementación del módulo, pero que consideramos un buen ejemplo para comentar este tipo de ampliación.

- Una buena ampliación para la ayuda semántica sería la de mejorar la ontología sobre la metáfora. Para ello valdría con aprovechar en mayor medida los recursos que aportan aplicaciones como Protégé para darle a la ontología capacidades como podría ser diferenciar cuando se le está preguntando por un concepto de tipo numérico, y en este caso referir la respuesta al de las constantes o variables según la decisión que la función de similitud considerará más adecuado. De esta manera se podrían solucionar preguntas realizadas en la fase de evaluación como por ejemplo *"Where can i find the 7"*. También se podría otorgar de capacidad

a la aplicación para solventar el caso en el que el número no venga escrito como una cifra sino como palabra *"Where can i find the seven"*. Y en ese caso resolver como antes.

- Si se realizara una mejora de la función de similitud evidentemente mejorarían los resultados obtenidos por la ayuda semántica.

- Como ya se comentó en la sección 2.9 una mejora muy interesante sería la de otorgarle a las ayudas de capacidad de ayuda proactiva, con esto queremos decir que este módulo mantuviera mediante máquinas de estados la situación actual del desarrollo del ejercicio que está siendo resuelto por el usuario así como la del supuesto desarrollo que debería seguir el usuario para resolver el ejercicio correctamente. Para así incluir la comparación entre el estado actual de la resolución del ejercicio y el supuesto estado en que se debería encontrar para ser resuelto correctamente, como parte de la toma de decisión de que respuesta devolver. De esta manera se podrían resolver preguntas del tipo *"What box must i need to pick now?"*, pregunta a la que ahora mismo respondería con el texto a cerca de las cajas (box).

3.8. Conclusiones

En este módulo se ha estudiado e implementado un sistema de ayuda semántica para un sistema de enseñanza interactivo. Se ha estudiado también la metáfora de Javy2, y se ha conseguido un sistema que de manera intuitiva y fácil responde a las dudas sobre dicha metáfora con bastante efectividad. Hemos aprendido que es necesaria una ayuda para el juego que trate del contexto y la metáfora.

Para la creación del módulo, primero hemos diseñado e implementado una ontología que trata de la metáfora de Javy2, así como del contexto y los controles. Una vez creada, hemos implementado en Java un sistema que seleccione de los nodos de la ontología aquellos conceptos que más se relacionen con un texto que se toma como entrada (la pregunta del usuario de Javy2), y devuelva la exlicación asociada al nodo sleccionada. También es capaz de devolver ayuda sobre un concepto parecido al anterior; es decir, dar más de un texto de ayuda para una misma

pregunta.

Hemos estudiado posibles ampliaciones, tales como una máquina de estados que proporcione ayuda proactiva sobre la resolución de los ejercicios o los objetivos parciales del juego; y hemos explicado cómo se podrían realizar ampliaciones o cambios sobre la ontología.

Este módulo se comparará con el módulo de ayuda semántica para ver cuál de los dos es el más adecuado a las necesidades de Javy2. El estudio comparativo entre ambos módulos se expondrá en el capítulo 4 resolviéndose cuál de los dos módulos se integra en Javy2.

Capítulo 4

Evaluación de las Ayudas Semántica y Sintáctica

La evaluación de las Ayudas, desarrolladas en la primera etapa del proyecto, se realizó en dos fases, una primera fue llevada a cabo por nosotros mismos, y la segunda por personal ajeno a la implementación. La razón de hacerlo de esta manera es que al realizar la primera fase de la evaluación nosotros mismos nos dimos cuenta de que las preguntas que hiciéramos, quisiéramos o no, estaban demasiado sesgadas, ya que tanto para la ayuda semántica como para la ayuda sintáctica nosotros sabemos que preguntas están capacitadas a responder y cuales no.

En el caso de la semántica es porque basa su búsqueda de soluciones en la lista de sinónimos asociados a cada concepto, dicha lista la hemos confeccionado nosotros, por lo que sabemos que palabras están y cuales no, y en el caso de que pensar en preguntas con palabras que no estuvieran en las listas simplemente pasarían a estarlo, por ello tuvo sentido que realizáramos la primera fase nosotros.

En el caso de la Ayuda Sintáctica, al basar su búsqueda de respuestas en la similitud entre la pregunta y los textos de ayuda, y al haber hecho también nosotros dichos textos, volvemos a encontrarnos con la misma situación sesgada.

Por ello recurrimos a un grupo de personas ajeno al desarrollo de las Ayudas pero con un conocimiento del juego aceptable para poder realizar preguntas adecuadas sobre Javy2. Este grupo se formó con 6

integrantes del grupo GAIA de la facultad de Informática de la Universidad Complutense de Madrid.

4.1. Evaluación realizada por nosotros

En esta primera fase se realizaron una serie de diez preguntas tanto al módulo de ayuda semántica como al de ayuda sintáctica y se anotaron los resultados obtenidos así como el tiempo tardado en devolver esta respuesta. Estos resultados nos sirvieron para afinar el tiempo de respuesta tanto de una como de otra y llegar a la conclusión de que era lo suficientemente eficiente en las dos versiones.

También concluimos que la ayuda semántica resolvía con mayor precisión las preguntas realizadas en función de las respuestas esperadas. De todas maneras estos resultados no nos parecieron fiables, ya que al realizar nosotros las preguntas sabíamos de antemano cuales iban a poder ser resueltas satisfactoriamente y cuales no. Lo cual aportaba mas interés si cabe a la segunda fase de evaluación, en la cual nosotros no tendríamos la responsabilidad de realizar las preguntas.

Hay que destacar que los módulos de ayuda pueden responder a preguntas relativas a la metáfora del juego y a preguntas relativas a la JVM. Sabiendo esto destacamos que en esta fase se limitaron las preguntas a la metáfora del juego y se excluyeron las preguntas relativas a la maquina virtual de Java. La razón es obvia si tenemos en cuenta que la ayuda semántica no dispone de información relativa a la JVM, cosa que la sintáctica sí, y ya que la intención de esta evaluación es la de decidir entre el uso de una de las dos para la integración con Javy2 dando por sentado que la parte que respecta a la JVM será resuelta por la ayuda sintáctica.

Las preguntas realizadas en esta primera fase de evaluación fueron las siguientes:

- **Preguntas realizadas:**

    1. Which is the key for moving right?

2. How can I move the camera?
3. How can I hunt?
4. How can I get the bytecode window?
5. How can I upgrade my level?
6. Who is the master that helps me with the load instruction?
7. Which is the security room?
8. Where are the enemies?
9. What is the elevator?
10. Which is the hot key to view the inventory?

- **Resultados obtenidos**

  Cuando decimos resultado esperado y obtenido nos referimos al documento *.txt que debería mostrar según la pregunta realizada, y al que muestra realmente.

  En ocasiones el nivel de similitud coincide en dos o más conceptos, en ese caso muestra el primero según el orden en que los encuentra. Esta situación nos llevo a la conclusión de la necesidad de mejorar la función de Similitud de la ayuda Semántica para que la búsqueda realizada fuera más precisa.

  Hay que destacar, a parte de lo ya comentado, que nos sirvió de ayuda para añadir nuevos sinónimos a los conceptos de la ontología, ya que el pararnos a pensar en preguntas posibles nos hizo encontrar nuevos sinónimos que durante la implementación no habíamos incluido en la ontología. Esto realmente sólo beneficia a la semántica, ya que la sintáctica utiliza un método distinto para resolver las cuestiones.

Siendo sinceros, esta primera fase de evaluación no nos aporto más conclusiones relevantes, el hecho de realizar nosotros las preguntas sesgó demasiado los resultados obtenidos como ya hemos comentado con anterioridad. En la figura 4.1 se pueden ver los resultados de esta fase.

- a.Sintáctico
- b.Semántico

Pregunta	Esperada	Obtenida	Tiempo
1-a	Movimiento_usuario	Movimiento_usuario	359ms
1-b		Movimiento_usuario	203ms
2-a	Camara	Movimiento_usuario	203ms
2-b		Camara	109ms
3-a	Caza	Caza	94ms
3-b		Caza	94ms
4-a	Como_Llegar_Bytecode	Teclas_Rapidas	140 ms
4-b		Como_Llegar_Bytecode	141ms
5-a	Niveles_Interior	Significado_Ascensor	94ms
5-b		Niveles_Interior	140ms
6-a	Interaccion_Mentores or TiposInfo	Experience	141ms
6-b		TiposInfo	156ms
7-a	Significado_Camara_Seguridad	Robo Camara Seguridad	94ms
7-b		Robo_Camara_Seguridad	141ms
8-a	Localizacion_Malos	Nada	141ms
8-b		Nada	31ms
9-a	Significado_Ascensor	Significado_Ascensor	94ms
9-b		Significado_Ascensor	110ms
10-a	Teclas_Rapidas	Como_llegar_inventorio	141ms
10-b		Teclas_Rapidas	110ms

Figura 4.1: Resultados primera fase evaluación

- Resultados Sintáctica: 3 aciertos sobre 10 preguntas, Tasa de Acierto=0.3 (30 %).
- Resultados Semántica: 8 aciertos sobre 10 preguntas, Tasa de Acierto=0.8 (80 %).

4.2. Evaluación por gente externa a la implementación

Tras la realización de esta primera fase de evaluación realizada por nosotros, llegó el momento de probar las Ayudas por personas ajenas al desarrollo de las mismas. Como comentábamos con anterioridad el grupo de evaluadores estaba formado por seis miembros de GAIA, los cuales tienen un conocimiento de Javy2 lo suficientemente amplio para poder realizar preguntas adecuadas para extraer resultados positivos de esta evaluación.

Lo que se hizo para desarrollar la evaluación fue lo siguiente. Se les proporcionó ejecutables de las dos versiones de la ayudas y se les pidió que realizaran las mismas preguntas a las dos, y una vez realizadas nos enviaran un documento donde se recogiera el grado de satisfacción obtenido con cada respuesta.

Antes de observar los resultados estadísticos que hemos recogido de las respuestas obtenidas por los evaluadores hay que tener en cuenta que estos han planteado las preguntas a las ayudas sin distinguir entre conocimiento de la JVM y de la metáfora, lo que hace que los resultados de la semántica haya que estudiarlos siendo conscientes de varios de los errores cometidos por ella han sido a causa de que se le realizaban preguntas para las cuales no tenía respuesta. En la sección de apéndices C se pueden consultar algunos de los archivos de respuesta obtenidos por los evaluadores:

- **Evaluador 1**
  - Realizadas 21 preguntas.
  - Semántica: 12 correctas, 9 incorrectas.
    - Tasa acierto: 0.57
    - Tasa fallo: 0.43
  - Sintáctica: No realizó preguntas.
- **Evaluador 2**

- Realizadas 19 preguntas.

- Semántica: 11 correctas, 8 incorrectas.
  - Tasa acierto: 0.58
  - Tasa fallo: 0.42
- Sintáctica: 8 correctas, 11 incorrectas.
  - Tasa acierto: 0.42
  - Tasa fallo: 0.58

- **Evaluador 3**
  - Realizadas 13 preguntas.

  - Semántica: 3 correctas, 10 incorrectas.
    - Tasa acierto: 0.23
    - Tasa fallo: 0.77
  - Sintáctica: No realizó preguntas.

- **Evaluador 4**
  - Realizadas 14 preguntas.

  - Semántica: 8 correctas, 6 incorrectas
    - Tasa acierto: 0.57
    - Tasa fallo: 0.43
  - Sintáctica: 6 correctas, 8 incorrectas.
    - Tasa acierto: 0.43
    - Tasa fallo: 0.57

- **Evaluador 5**
  - Realizadas 17 preguntas.

  - Semántica: 6 correctas, 11 incorrectas.
    - Tasa acierto: 0.35
    - Tasa fallo: 0.65
  - Sintáctica: 6 correctas, 11 incorrectas.

    - Tasa acierto: 0.35
    - Tasa fallo: 0.65

- **Evaluador 6**
    - Realizadas 20 preguntas.
    - Semántica: 14 correctas, 6 incorrectas.
        - Tasa acierto: 0.7
        - Tasa fallo: 0.3
    - Sintáctica: 11 correctas, 9 incorrectas.
        - Tasa acierto: 0.55
        - Tasa fallo: 0.45

- **Cálculo total**
    - Semántica: Sobre 104 preguntas, 54 correctas, 50 incorrectas.
        - Tasa acierto: 0.52 (52 %)
        - Tasa fallo: 0.48 (48 %)
    - Sintáctica: Sobre 70 preguntas, 31 correctas, 39 incorrectas.
        - Tasa acierto: 0.44 (44 %)
        - Tasa fallo: 0.56 (56 %)

Como comentábamos antes de mostrar las estadísticas, en el caso de la semántica al trabajar con el sesgo de no poseer conocimiento sobre la JVM, podríamos recalcular sus resultados de la siguiente manera. Si excluimos del conjunto de preguntas realizadas las que se refieren a la JVM, tendríamos que se le realizaron 16 preguntas sobre la JVM. Por tanto:

- Sobre 88 preguntas obtuvo 54 correctas, 34 incorrectas.
    - Tasa acierto: 0.61 (61 %)
    - Tasa fallo: 0.39 (39 %)

Lo más importante de este recálculo de los resultados de la semántica no es la mejora obtenida (del 9 %) sino que son resultados más reales en cuanto a las preguntas a las que ha tenido que enfrentarse.

4.3. Conclusiones obtenidas de la Evaluación

- Varias de las palabras utilizadas por los evaluadores no habían sido tenidas en cuenta al realizar la ontología, por ello esta evaluación permite que se complete dicha ontología. Esto es una de las razones por lo que era necesario que estas pruebas fueran realizadas por gente ajena al desarrollo. Esto es una gran ventaja de la ayuda semántica ya que cuanto más y por más gente distinta sea probada más riqueza de palabras se obtendrá. Un claro ejemplo de lo comentado en este punto es el sentido otorgado a la palabra *"Where"*, nosotros inicialmente la filtrábamos en el ciclo inicial de análisis de la pregunta, pero la evaluación nos ha hecho ver lo conveniente de añadirla en la ontología en los conceptos que se refieran a localizaciones. Por ejemplo a la pregunta *"where are the boxes?"* la palabra *where* le concedería más peso a la respuesta adecuada (Localización de las cajas) que a una posible respuesta como podría ser descripción de las cajas. Otras palabras como esta han sido incluidas en la ontología una vez realizada esta evaluación. Como resultado de ello se ha obtenido la versión 4.0 de la ontología, que hasta que se realicen nuevas evaluaciones o cambios en la metáfora, será la que esté integrada en el Javy2.

- Observando las preguntas realizadas se ve claramente que al realizar preguntas sobre la JVM el módulo de ayuda semántica no obtiene respuesta y el sintáctico quizás devuelva una demasiado larga y técnica. La respuesta del módulo semántico es lógica ya que no se le ha proporcionado conocimiento acerca de la JVM, pero al ver el sentido que los evaluadores daban a sus preguntas, se entiende que lo que buscaban era obtener una repuesta que relacionara a la metáfora con la JVM. El módulo semántico sí tiene respuestas que cumplan este menester, todas las partes de la JVM representadas metafóricamente en el juego tienen su explicación en el módulo. Por ejemplo la pila, esta es una parte de la JVM que está representada en la metáfora, si se le pregunta al módulo semántico *"Where is the heap"* o *"What represents the heap in*

the game" le reponderá adecuadamente, pero si lo que se le pregunta es *"What is the instruction to make a push on the heap"*, el módulo semántico no responderá adecuadamente, a diferencia del sintáctico. La solución para este conflicto parece ser añadir a los conceptos de la metáfora en la ontología, conceptos relacionados con la JVM, aunque esto puede ser arriesgado ya que en el momento de juntar la Ayuda Sintáctica sobre la JVM con la ayuda Semántica sobre la metáfora podría provocar cierto conflicto entre las respuestas obtenidas. Solo resoluble comprendiendo el sentido concreto de la pregunta realizada.

- Se observan en la evaluación preguntas usando números como el 5 o el 6. En ellas el evaluador espera obtener una respuesta relacionada con el concepto de las constantes, por ejemplo *"Where can i find the 7"*. Semánticamente este el sentido que tienen estos números. La Ayuda Sintáctica no puede resolver esta pregunta satisfactoriamente, ya que la búsqueda se realiza sobre la similitud entre la pregunta y la base de casos de textos a devolver, en el caso de la Ayuda Semántica la solución para resolver esto sería usar recursos de las herramientas de OWL, los cuales permiten distinguir cuando un concepto a añadir se refiere a un tipo numérico. Esto facilitaría la labor si se identificaran a dichos tipos como instancias de constantes lo que relacionaría la respuesta obtenida con el texto de descripción perteneciente a dichas constantes. Esta tarea se ha dejado como posible ampliación para más adelante y viene tratada en la sección 3.7.

- Otras preguntas realizadas en la evaluación esperan obtener una respuesta que sea "YES" o "NO". Cosa del todo imposible para la aplicación, ya que desde un principio se diseñó como ayuda textual y no como un razonador que proporcione respuesta razonando, nunca mejor dicho, sobre la pregunta realizada. Un ejemplo de este caso es la pregunta realizada por una de los evaluadores: *"Is that an elevator?"*, tanto la semántica como la sintáctica relacionaron su respuesta con el ascensor, repuesta que se puede considerar satisfactoria conociendo las limitaciones existentes en los dos sistemas. Sería interesante capacitar a las ayudas de conocimiento interactivo sobre a que se refiere el usuario. Lo que podría capacitar al sistema de conocimiento suficiente para responder a la pregunta citada y así resolver la respuesta en tiempo de ejecución, consulte las secciones 2.9 y 3.7.

- Una clara conclusión obtenida viene dada por el proceso de constante evolución y desarrollo que está sufriendo tanto el Javy2 como su metáfora. Esto provoca que existan preguntas que la aplicación no sepa responder al referirse a conceptos, que cuando se inició la creación de las ayudas, no existían. Esto al menos tiene fácil solución y se limita a realizar ampliaciones y modificaciones en la ontología así como en los textos de la base de casos. Para más información al respecto consulte la sección 3.7.

El proceso de evaluación tiene como objetivo mejorar los módulos de ayuda, identificando tanto los puntos débiles como los fuertes. Nosotros seguimos el siguiente proceso de mejora.

Se analizaron las preguntas realizadas por los evaluadores, y ante los resultados devueltos que no les satisfagan, se buscaron las palabras claves introducidas en dichas preguntas y con ellas y la respuesta esperada por el evaluador, se modificó la ontología de la Ayuda Semántica.

Como conclusión final llegamos a que la Ayuda que sale victoriosa para ser incluida en el juego para solventar las preguntas referidas a la metáfora es la Ayuda Semántica. Por tanto el juego dispondrá de Ayuda Sintáctica cuando las preguntas se refieran a conceptos de la JVM, y Semántica cuando estas sean referidas a la metáfora. Esto soluciona el problema planteado como primera conclusión en esta sección y nos hace pensar que de esta manera aprovecharemos al máximo los recursos de las ayudas.

Capítulo 5

Recomendador de ejercicios

5.1. Objetivo y motivación

Como ya hemos comentado, el juego Javy2 se está creando con la motivación principal de que sus usuarios aprendan el manejo de la Máquina Virtual de Java y su lenguaje, el **bytecode**. El sistema Javy2 propone al jugador un ejercicio que debe resolver. Para ello hemos creado un módulo que recomienda el ejercicio más apropiado para cada alumno o usuario, en función de su conocimiento y destreza, descritos éstos por el **perfil de usuario**. Para ésto hemos utilizado un sistema CBR.

Case-Based reasoning o CBR (Razonamiento basado en casos) es el proceso de solucionar nuevos problemas basándose en las soluciones de problemas anteriores. El Case-Based reasoning (CBR) es una manera de razonar haciendo analogías. Hemos considerado muy útil diseñar nuestro recomendador como CBR, basándonos en una base de casos formada por ejercicios Java.

El objetivo final es que el alumno vaya practicando conceptos que no tiene todavía claros y empiece a familiarizarse con aquellos que aún no conoce, siempre y cuando se encuentre ya preparado para ello. Para esto es necesario evitar la monotonía que significan los ejercicios demasiado fáciles para cada nivel, sin aumentar demasiado la dificultad.

El ejercicio será más apropiado cuanto más de ellos tengamos para

comparar, de forma que necesitamos también una gran base de ejercicios organizados por sus conceptos de compilación. Y para contar con ella, sería extremadamente útil una herramienta que nos permitiera manipular ejercicios en Java y convertirlos en código OWL de forma automática; para que el esfuerzo humano se limitase a la selección de dichos ejercicios. De esta idea nos surgió la continuación de un submódulo que estaba desarrollando en el departamento de GAIA Pedro Pablo Gómez Martín; que transforma código Java en OWL (sección 5.4). Será de este código OWL del que se seleccionarán los ejercicios adecuados, como se puede ver en el apartado 5.7.

Una vez que un ejercicio haya sido elegido, un módulo diferente, el **Generador de Escenarios**, se encargará de adaptarlo para convertirse en una pantalla de Javy2. Esto se ve en el capítulo 6.

5.2. Estructura general

En este apartado se explicarán a grandes rasgos la estructura del módulo completo, así como una breve descripción de los diferentes submódulos, junto con las relaciones entre ellos. Todo esto se irá explicando más detalladamente durante el resto del documento.

El sistema se compondrá de los siguientes módulos (ver figura 5.1):

- **Editor de perfiles de usuario.** Para que un usuario externo pueda acceder fácilmente a los perfiles de usuario para crear o modificarlos, mientras que el módulo general no se encuentre integrado en Javy2.
- **Manejador de perfiles de usuario.** Se encarga del tratamiento de los perfiles de usuario como objetos y en formato XML. Trata las inserciones y recuperaciones de perfiles de usuario en XML, ya que será en XML como se guardarán dichos perfiles. Recibe las órdenes del **Editor de Perfiles** y del **Recomendador**, y les proporciona la información necesaria sobre dichos perfiles.
- **Herramienta Java2OWL**, elaborada para automatizar la conversión de los ejercicios, seleccionados para formar parte de la base

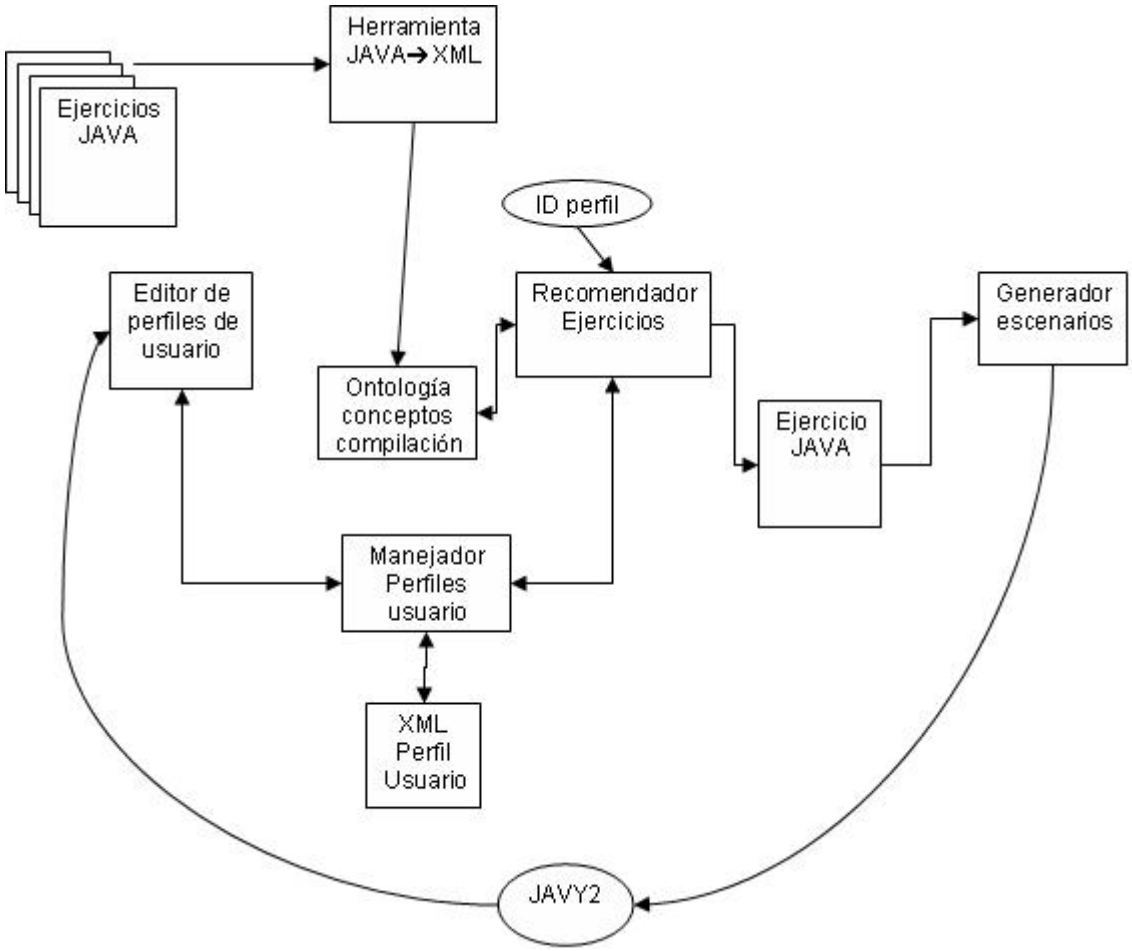

Figura 5.1: Estructura general del recomendador de ejercicios.

de ejercicios del Javy2, al formato OWL. Este formato nos permite introducir a los ejercicios como instancias de una ontología sobre conceptos de Java. Este módulo, que ya estaba comenzado, nos ha resultado muy importante para conseguir una buena base de ejercicios Java en formato OWL, para mejorar las futuras selecciones.

- **Recomendador de ejercicios.** Módulo principal. Elige el ejercicio más apropiado de la base de conocimiento para cada perfil. Consta de un sub-módulo que se encarga de la inserción, recuperación y borrado de los perfiles como código OWL. El perfil es insertado como un individuo en la **ontología sobre conceptos de compilación**[1], con sus relaciones, como se explicará en

[1]Esta ontología es un proyecto realizado por Pablo Palmier y, aunque todavía está en

el apartado 5.7.1.

Por una parte, en un preciclo del recomendador, la herramienta Java2OWL[2], que transforma código Java en instancias de la ontología sobre conceptos de compilación y relaciones entre ellas, recibe la base de casos de ejercicios en Java y genera el código OWL, introduciendo los conceptos de cada ejercicio en la ontología como individuos de la misma.

Una vez hecho esto, desde el editor de perfiles el usuario podrá añadir o modificar perfiles en el fichero XML de perfiles emitiendo las órdenes correspondientes al Manejador de perfiles.

El recomendador de ejercicios le pide al manejador de perfiles la información de un determinado perfil, que se encuentra en un fichero XML. Este perfil se incluye por un submódulo del recomendador en la ontología, y una vez introducido en ella, mediante una función de similitud y unos algoritmos de recuperación tanto computacional como representacional, escoge el ejercicio más apropiado para ese perfil.

El ejercicio elegido, en Java, es pasado al Generador de Escenarios que crea un fichero XML con la información del nuevo escenario correspondiente al ejercicio. Esta información es leída por Javy2, que a su vez será el que envíe la información de actualización de perfil.

5.3. Obtención de Base de Ejercicios Java

Una parte muy importante dentro del recomendador de ejercicios es evidentemente la base de ejercicios sobre la cual se va a realizar la recomendación. Cuanto más amplia sea más variedad de retos se le podrá proponer al alumno (usuario de Javy2). La primera versión de esta base también ha sido realizada por nosotros.

---

sus comienzos, pretende representar jerárquicamente todos los conceptos de compilación del lenguaje Java.

[2]El módulo Java2OWL convierte ejercicios en Java en código OWL. Su finalidad es la de obtener una buena base de conocimiento (base de ejercicios) para poder obtener de ella aquél que se adapte mejor a las necesidades del usuario. Esta herramienta está explicada en profundidad en el capítulo 5.4

La recolección de estos ejercicios tuvo como inicio el uso de una lista de ejercicios de tipo imperativo que ya habían sido recolectados por Pedro Pablo y Marco Antonio Gómez Martín. Esta lista estaba formada por veinte ejercicios sencillos de tipo imperativo, por lo tanto nuestro objetivo consistía en ordenar por orden de complejidad estos veinte, añadir nuevos ejercicios imperativos a la lista y recolectar un número no determinado de ejercicios Java que utilizaran orientación de objetos. Como resultado final obtuvimos 28 ejercicios de tipo imperativo y 22 orientados a objetos. Para recolectar estos últimos nos pusimos en contacto con profesores de la asignatura *Programación Orientada a Objetos* de la facultad de Informática de la Universidad Complutense de Madrid, los cuales nos proporcionaron ejercicios solucionados por sus alumnos del tipo requerido. Esto hizo que tuviéramos que realizar una fase de revisión de estos ejercicios para poder corregirlos y adaptarlos a las necesidades del Recomendador de Ejercicios de Javy2.

En los dos cuadros siguientes 5.1 y 5.2 se muestra el nombre de los ejercicios ordenados por complejidad, tanto los de tipo imperativo como los orientados a objetos.

Del 1 al 14	Del 15 al 28
DeclareInt	IfArray
DeclareLong	ForArray
DeclareFloat	For3
DeclareDouble	Fibonacci
DeclareBool	SumaPares
DeclareInt2	MasSumaPares
DeclareObject	NumerosDeDigitos
Declare	CalcularPi
Arithmetic1	EsNumeroPrimo
Arithmetic2	Ordena
Arithmetic3	OrdenaArray
IfFalse	ComprobarTriangulo
IfTrue	CalcularVuelta
For	FechaValida

Cuadro 5.1: Ejercicios Imperativos ordenados por complejidad

Queda como tarea posterior ampliar lo máximo posible la base de ejercicios tanto con ejercicios de tipo imperativo como de tipo orientado a objetos.

Del 1 al 11	Del 12 al 22
Monedero	Rect
Suma	Fraccion
ConcatenacionString	MasConjuntoDeFiguras
Trio	Buzon
ColaImp	Persona
Lista	Banco
ConjuntoDeEnteros	Banco_v2
Primos	Banco_v3
Primos_V2	CatalogoArtistas
ConjuntodeFiguras	CatalogoArtistas_v2
ConjuntodeFiguras_v2	Prado

Cuadro 5.2: Ejercicios Orientados a objetos ordenados por complejidad

5.4. Módulo Herramienta Java2Owl

Este módulo se dedica a la explicación detallada del diseño, desarrollo y funcionamiento de la herramienta Java2Owl. Ésta ha sido elaborada para automatizar la conversión de los ejercicios, seleccionados para formar parte de la base de ejercicios del Javy2, al formato OWL. Este formato nos permite introducir a los ejercicios como instancias de una ontología sobre conceptos de Java.

Primero explicaremos la motivación de la herramienta así como el objetivo que se persigue con su creación. Acto seguido trataremos cómo adquirimos el conocimiento necesario y a continuación pasaremos a explicar su estructura de clases y funcionamiento, para ello usaremos un ejemplo práctico de la conversión de tres ejercicios de tipo imperativo. Para terminar explicaremos qué queda como ampliación para más adelante, y como realizar dicha ampliación.

5.4.1. Objetivo y motivación

La necesidad de este módulo se nos planteó por parte de Belén Díaz en el momento en el que comenzábamos el desarrollo del módulo recomendador de ejercicios. Pretendíamos realizar un módulo que utilizando una amplia base de ejercicios y un perfil del usuario decidiera qué ejercicio era el más conveniente para que éste realizara. En este punto Belén

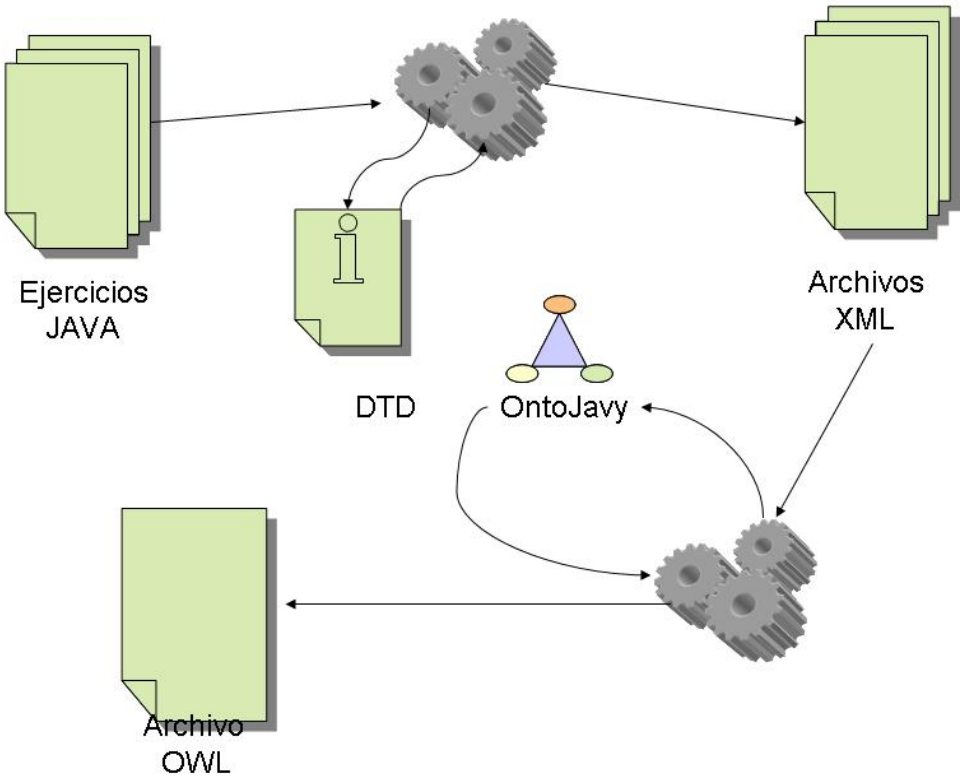

Figura 5.2: Diseño Módulo Java2Owl

nos hizo ver que la mejor manera de realizar este proceso de decisión era mediante el uso de ontologías, pero para ello requeríamos que los ejercicios estuvieran en un formato distinto al fuente, el OWL, formato de las ontologías.

El diseño de esta herramienta se le había encargado a Pedro Pablo Gómez-Martín[3]. Una vez diseñado y elaborado un esqueleto inicial se nos encargó la tarea de su desarrollo.

A grandes rasgos lo que hace la herramienta es recibir archivos Java, los cuales transforma a XML mediante un parser y siguiendo una DTD definida para Java. Una vez realizado este paso se utiliza una ontología llamada Ontojavy (Pablo Palmier[4]) para convertir los ejercicios de XML a OWL.

[3]Integrante de GAIA y responsable del diseño de la Herramienta
[4]colaborador de GAIA, y desarrollador de la OntoJavy

5.4.2. Adquisición general de conocimiento

El que esta herramienta fuera una tarea asignada en la que el diseño ya estaba realizado e incluso se nos proporcionaba un esqueleto inicial bastante detallado, facilitaba el proceso de adquisición de conocimiento.

Lo primero que hicimos fue reunirnos con Pedro Pablo Gómez-Martín (encargado del diseño) para que nos explicara qué es lo que tenía que hacer el módulo y cómo estaba diseñado para tal cometido. Para ello nos proporcionó un documento explicando clase a clase cuáles deberían ser sus responsabilidades.

Una vez entendido el propósito de la herramienta pasamos a estudiar en profundidad los distintos pasos que ésta realizaría para llegar al propósito final. En la figura 5.2 se representa el proceso seguido por el módulo.

El primer paso, encargado de transformar los ejercicios Java a XML es realizado por el parser Java2xml elaborado por Harsh Jain[5], al realizar pruebas para familiarizarnos con el parser, caímos en la cuenta de que nos podía traer problemas partes de la ejecución que realizaba, así que con la colaboración de Pedro Pablo realizamos algún retoque para que el funcionamiento se adaptara a nuestras necesidades. Por ejemplo el parser original nos daba problemas con los operadores lógicos.

En cuanto a la adquisición de conocimiento la parte más importante de este paso fue el estudio en profundidad de la DTD que este sistema utiliza para generar los XML, ya que en función de dicha DTD, iba a apoyarse el desarrollo de la aplicación. Por ello le dedicamos un tiempo extenso a observar las distintas elementos, entidades y atributos recogidos.

Una vez comprendido el árbol de la DTD pasamos a una fase de elección de elementos que íbamos a implementar en el desarrollo de la herramienta. Para ello dependíamos de la Ontología (OntoJavy), ya que solo podríamos crear instancias dentro de dicha ontología de los elementos que estuvieran recogidos como conceptos en Ontojavy. Al ser el encargado del desarrollo de la ontología un colaborador de GAIA (Pablo Palmier) acordamos con él, Belén y Pedro Pablo la necesidad

[5]Department of Computer Science and Engineering, IIT Bombay

de tener cuanto antes la parte más baja de la ontología, es decir tener cuanto antes las hojas de conceptos, y dentro del gran número de hojas existentes, le pedimos que realizara primero las correspondientes a elementos de la DTD que fueran necesarios para transformar en OWL, ejercicios de tipo imperativo, dejando los ejercicios orientados a objetos para una posible ampliación posterior. Por ello en esta fase de adquisición de conocimiento concluimos la necesidad de disponer de conceptos en la ontología para las entidades de la DTD mostradas en el cuadro 5.3.

Entidad	Entidad	Entidad
SouceProgram	Unary-expr	VarRef
Class	If	New-array
Literal-null	Array-ref	Break
Literal-string	Type	Case
Literal-char	type-argument	Default-Case
Literal-number	Type-parameters	ArrayInitializer
Literal-boolean	Local-variable	dimExpr
Expr	Method	Switch
Binary-expr	Formal-Arguments	Paren
Lvalue	Block	switch-block
Assignement-expr	Loop	base

Cuadro 5.3: Entidades de la DTD implementadas en la Herramienta

Aparte de estos elementos también se pidió el desarrollo, dentro de la ontología, de los que de ellos descendieran en el árbol de la DTD. Para consultarlos con más profundidad se recomienda acudir al Apéndice D, allí se encuentran la declaración de estos elementos y sus descendientes tal y como aparecen en la DTD.

Una vez recibimos una primera versión de Ontojavy con unos cuantos elementos de la lista, y estudiado su estructura y propiedades, ya estábamos listos para comenzar con el desarrollo del módulo.

5.4.3. Estructura de clases

En esta sección pretendemos explicar la estructura de la herramienta y la responsabilidad asignada a cada clase. También explicaremos el funcionamiento general del Módulo.

Antes de entrar en detalle con la funcionalidad de cada clase vamos a exponer unos pocos aspectos técnicos del módulo Java2Owl, éstos nos servirán para explicar en mayor profundidad su funcionamiento. Estos aspectos están extraídos del pequeño documento de diseño que nos proporcionó Pedro Pablo durante la fase de adquisición de conocimiento:

- La herramienta utiliza el conversor de Java a XML directamente invocándolo como a cualquier otro método Java. Eso está implementado en la clase Java2XML, y hace uso de java2xml-1.0.jar.

- La herramienta utiliza el analizador de XML nativo del JDK. El análisis(obviamente parcial) se realiza en la clase XML2owl. La idea es dividir el análisis en diferentes métodos, seguramente uno por cada elemento de la DTD original del XML.

- La herramienta utiliza Jena[6] para grabar el owl. Toda la "ontologíaresultante se crea en memoria utilizando la clase auxiliar JavaOntology, que envuelve a Jena ahorrando los detalles sucios a la clase XML2owl comentada anteriormente.

- El programa principal hace un análisis relativamente sofisticado de los parámetros proporcionados por el usuario, dando varias opciones para generar la salida. Para eso utiliza la librería *jargs*[7].

- A lo largo de todo el programa se hace uso de log4j[8] para generar información sobre el proceso. Por parámetros es posible especificar cuanta de esa información queremos ver por pantalla.

La clase Main

Esta es la clase principal del programa. Es la encargada de tratar los parámetros de entrada y en función de ellos decidir qué hacer en cada caso. Las posibilidades para las que está preparada la herramienta según las opciones que se le activen por parámetro son las siguientes:

[6] Jena es un framework desarrollado por HP Labs para manipular metadatos desde una aplicación Java

[7] Librería Java para control de los atributos de entrada

[8] Librería Java para gestionar las salidas de los logs(mensajes de estado o información)

- Puede recibir uno o varios archivos Java a la vez para ser tratados.
- Puede conservar o no el archivo XML intermedio generado.
- Puede crear un nuevo archivo OWL de salida o sobreescribir uno ya existente, pudiendo elegir también si se quieren añadir las nuevas instancias a las ya existentes o si se quiere sobreescribir por completo.

Un ejemplo de entrada de parámetros válido para esta clase sería:

-f Salida.owl -n Arithmetic1.java

Con la opción -f indicamos que fichero de salida queremos que utilice, con la opción -n indicamos que queremos que el archivo OWL se sobre escriba por completo.

La Clase Java2XML

Recibe una serie de nombres de fichero con código Java, los interpreta todos sintácticamente asumiendo que forman en conjunto un programa, y genera un XML con el árbol sintáctico de todos ellos. Se basa completamente en el programa java2xml de Harsh Jain, y de hecho lo utiliza. No obstante, en lugar de invocar a dicho programa utilizando invocaciones a nuevos procesos, lo que hacemos es utilizarlo como una librería e invocarlo desde Java como se utilizaría cualquier otra clase.

Como comentábamos anteriormente en esta clase se han tenido que hacer un par de cambios para adaptarlo a nuestras necesidades, por ejemplo el programa original ponía la salida siempre por defecto como *output.xml* así que se ha modificado esta opción para poder poner el nombre del archivo nosotros, si esa es la opción elegida mediante la clase Main. También se ha retocado el código para solucionar ciertos problemas que surgían con los operadores lógicos. Para todo esto nos intentamos poner en contacto con el autor del programa original Harsh Jain, pero no respondió a nuestros mensajes. Pero como el programa dispone de licencia CCPL[9], la cual permite el uso y modificación del

[9]Creative Commons Public License

código, siempre y cuando se redistribuyan los cambios y se nombre al autor original, aclaramos en este punto que los cambios realizados se han mantenido como un módulo externo al resto del trabajo, ya que si lo hubiéramos integrado tendríamos que someter todo el trabajo de esta parte bajo la Licencia CCPL.

Esta clase por tanto se encarga de generar el XML mediante el uso de la DTD comentada con anterioridad. Una vez realizado esto entra en juego la clase Xml2Owl.

La Clases XmlAux y Xml2Owl

XmlAux tiene una función muy clara, la de proporcionar de métodos auxiliares para inspeccionar nodos DOM[10] para obtener los valores de los atributos que se le indiquen, o el conjunto de los nodos hijos llamados de una determinada manera sobre un nodo DOM padre.

Estos métodos que proporciona la clase XmlAux son usados por la siguiente clase que pasamos a explicar, la clase Xml2Owl, la cual se puede considerar una de las dos claves del proceso de conversión Java→OWL.

Para hacer una introducción a lo que hace la clase Xml2Owl mostramos lo que el Javadoc de dicha clase dice:

"Clase que recibe como parámetro el nombre de un fichero XML con el árbol sintáctico de un programa Java y guarda todo el código en un fichero .owl que sigue la ontología de Java. El único método interesante para el usuario es xml2owl(...), que recibe el nombre de un fichero con el XML de un conjunto de ficheros Java analizados y convertidos, y escribe en otro fichero el mismo código pero en formato OWL siguiendo la ontología base de Java. Si se produce algún problema, este método devuelve falso.Es posible obtener información textual sobre la causa del error invocando a getLastError (). A nivel de implementación, durante el procesado del XML se hace uso del atributo _ontology para ir construyendo el ejercicio en owl. Esto evita tener que pasar continuamente la ontología a medio construir como parámetro a los múltiples métodos

[10]Forma de representar los elementos de un XML como objetos que tienen sus propios métodos y propiedades

auxiliares."

Como decimos en el párrafo anterior las dos claves de este clase son el método *xml2owl* y el atributo *_ontology*. El método es el encargado de comenzar la inspección del árbol DOM desde el nodo inicial, a partir de esa invocación se van encadenando llamadas hacia abajo del árbol a los distintos métodos encargados de tratar los distintos tipos de posibles nodos de la DTD. Estos métodos nombrados con el formato *parseNombreElementoNode()* son los encargados de tratar los distintos atributos de sus respectivos nodos, continuar el descenso en el árbol de nodos con los que les son inmediatamete inferiores, y lo más importante para el proceso, utilizar al atributo *_ontology* de clase *JavaOntology* para invocar al método correspondiente al nodo que trata, encargado de, como explicaremos a continuación, crear e introducir la instancia correspondiente en la OntoJavy.

Cuando el proceso de descenso del árbol de nodos finaliza, se devuelve el control al método principal *xml2owl* que en caso de que localice algún error lo reporta y devuelve falso y si no devuelve cierto.

Comentamos en este momento, aunque volveremos sobre ello en la sección Trabajo Futuro 5.4.5, que la implementación desarrollada en esta clase es la correspondiente a los métodos necesarios para convertir cualquier ejercicio Java de tipo imperativo a OWL. Pero destacamos que ya que durante el desarrollo de este módulo solo recibimos dos versiones de la OntoJavy, las cuales no venían provistas con el número necesario de conceptos para cubrir todos los ejercicios imperativos, supuso que parte del código encargado de varios de estos nodos esté realizado convenientemente para la creación de las instancias, pero comentado en la parte que se maneja al atributo *_ontology* para evitar conflictos. En el momento que se evolucione en la OntoJavy solo será necesario buscar las partes comentadas de los métodos correspondientes, etiquetadas con ToDo, para decomentarlas y probar su funcionamiento, esto se abordará con más detalle en la sección Trabajo Futuro 5.4.5.

La Clase JavaOntology

Es la encargada de enmascarar todo el proceso que rodea al uso de la OntoJavy para crear e introducir instancias, recurrimos de nuevo al

Javadoc de la clase para introducir su explicación:

Clase que oculta los detalles sucios de la creación de un ejercicio en formato owl.En el constructor recibe la ruta local en la que se encuentra el fichero owl con la ontología base donde están definidos los conceptos de los que se crean instancias para la definición del ejercicio. Si no se especifica ninguno, se asume la ruta mantenida en la constante javaOntologyFile. La clase requiere inicialización en dos pasos, por lo que tras la invocación al constructor es necesario invocar a createBasicExerciseOWL(), que podría, de hecho, fallar. La invocación a este método borra el anterior o añade el ejercicio al ya existente. Por último, se dispone de métodos para añadir elementos al ejercicio que se está construyendo (clases, etcétera), así como uno para volcar a un fichero el resultado en un fichero owl .

Esta clase tiene una constante por cada concepto y propiedad existente en la OntoJavy, así como constantes para mantener el espacio de nombres de la misma y para mantener el nombre del ejercicio que se está tratando en cada momento. Tiene los métodos necesarios para crear una instancia, una propiedad y demás cosas necesarias para la gestión de la ontología. Estos métodos se encargan de enmascarar el uso de la librería Jena.

El resto de métodos son los encargados de tratar a cada nodo de la DTD por separado, su funcionamiento en todos los casos es el siguiente. Reciben los atributos correspondientes de cada nodo, si es que los tienen, y crean la instancia concreta que les toca a cada uno, con las propiedades que sean necesarias en cada caso, el formato del nombre del método es *createNewNombreConcepto*. Las instancias las incluyen en la ontología con el nombre del ejercicio al que corresponden más el nombre de cada tipo de instancia particular y por último el número de instancia que es dentro del ejercio al que pertenece. La razón de añadir este número es para diferenciar dos instancias del mismo tipo de concepto dentro de un mismo ejercicio, si no incluyéramos el número de instancia que es se sobrescribirían las instancias del mismo tipo dentro del mismo ejercicio.

Como decíamos con anterioridad, estos métodos son invocados por la clase *xml2owl* en el método corresponda entre el nodo y el concepto.

El problema que comentábamos antes a cerca de lo corta que se nos quedó la versión de la que dispusimos de la ontología, también afecta a esta clase. El código está probado y funciona para los conceptos de los que disponíamos en la OntoJavy, en el caso del resto (para los ejercicios imperativos), está desarrollado pero no hemos podido probar su correcto funcionamiento. De nuevo se ha dejado lo más comentado posible mediante etiquetas ToDo donde decomentar trozos de código para que funcione, aunque es posible que haya métodos que haya que adaptar según la forma y propiedades que tomen finalmente en la siguiente versión de la OntoJavy. De todas maneras más adelante en la sección Trabajo Futuro se trata esta cuestión con detalle.

5.4.4. Funcionamiento y ejemplos (Validación)

El funcionamiento del módulo Java2Owl ya ha quedado más o menos claro con la descripción de clases de la sección anterior. Ahora nos disponemos a detallar con más profundidad la traza de ejecución de la herramienta, para ello mostramos en un principio un diagrama de secuencia con el proceso que sigue la herramienta durante la ejecución, para luego mostrar un ejemplo real sobre tres ejercicios Java en el que mostraremos el resultado obtenido con varias imágenes del estado de las instancias en la ontología.

Diagrama de secuencia de la ejecución de la Herramienta

En la figura 5.3 mostramos el diagrama de ejecución de un ejercicio en el que se genera el archivo OWL sin problema.

1. Java2xml.createxml(...)
2. Xml2owl.xml2owl(...)
3. XmlAux.getChilds(...)
4. _ontology.createNewClass(...)
5. fin de getJavaSourceProgramNode
6. System.out.println("Todo ha ido bien")

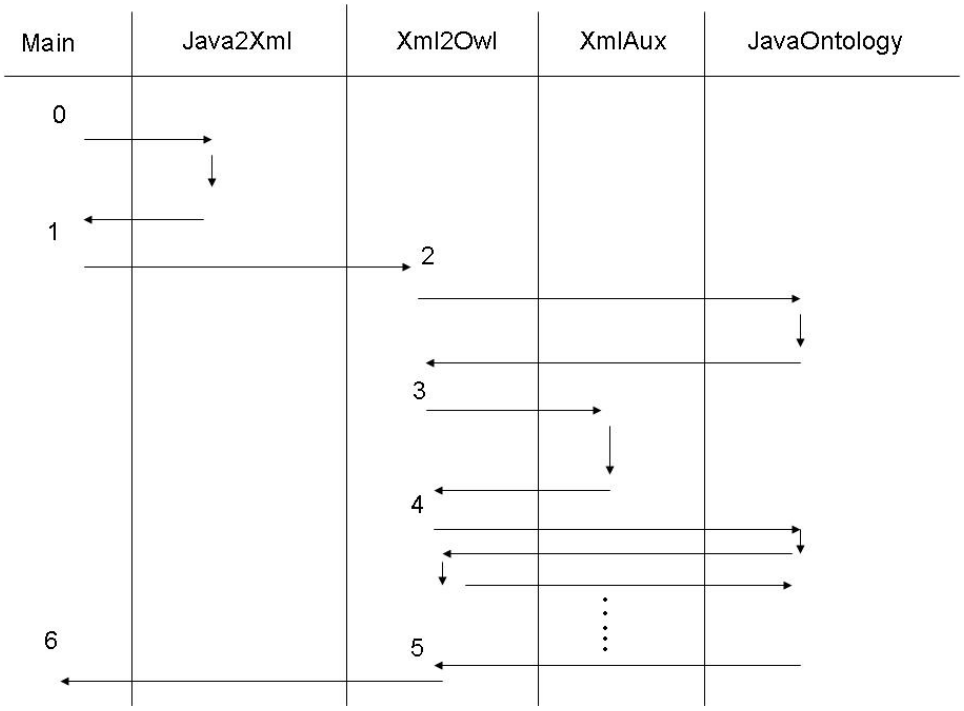

Figura 5.3: Diagrama de secuencia Herramienta

Entre el punto 4 y el 5 se realizan tantas llamadas entre la clase Xml2Owl y JavaOntology como nodos tenga el árbol sintáctico del ejercicio.

Ejemplo de funcionamiento, validación

El ejemplo que vamos a exponer en esta sección consiste en la utilización de la herramienta para la creación de un archivo OWL a partir de tres ejercicios Java. Para explicar el proceso con más detalle haremos la ejecución de cada ejercicio por separado en lugar de realizarla con los tres a la vez, opción para la que el módulo está preparada.

Primero indicaremos los parámetros de entrada necesarios para realizar la ejecución, mostraremos el código fuente de cada ejercicio, el código XML intermedio que se genera y por último mostraremos una captura de pantalla de como va evolucionando las instancias de la ontología a medida que se va ejecutando la transformación de cada ejercicio.

Teniendo en cuenta las limitaciones ya comentadas acerca de la versión utilizada de la ontología, los ejercicios que vamos a transformar van a ser tres de los más sencillos del conjunto de ejercicios recolectadas para ser utilizados en el Javy2. Veamos uno por uno su código y su como evoluciona el archivo owl de salida:

Ejecución del ejercicio Java Arithmetic1 sobre Java2Owl

Para transformar este ejercicio a OWL los parámetros de entrada necesarios son los siguientes, téngase en cuenta que el archivo de salida se genera por primera vez en esta ocasión y que queremos que se conserve el xml intermedio para poder mostrarlo:

Parámetros: -f Ejemplo.owl -k intermedioXml.xml -n Arithmetic1.java

La opción f asocia el nombre de ejemplo.owl a la salida, la opción k le hace mantener el xml con el nombre intermedioXml y la opción -n le indica que el archivo de salida se crea por primera vez.

```
class main {
        public static void main (String [] args ) {
                int c = 1;
                int a = 7;
                int b = c + a - 3;
        }
}
```

Arithmetic1.java

El archivo xml intermedio es el siguiente:

```
<!-- Generated by Java2XML http://java2xml.dev.java.net/ -->
<java-source-program>
    <java-class-file name="Arithmetic1.java">
        <class name="main" visibility="protected">
            <method name="main" visibility="public" static="true">
                <type name="void" primitive="true"/>
                <formal-arguments>
                    <formal-argument name="args">
                        <type name="String" dimensions="1"/>
                    </formal-argument>
                </formal-arguments>
                <block>
                    <local-variable name="c">
                        <type primitive="true" name="int"/>
                        <literal-number kind="integer" value="1"/>
                    </local-variable>
                    <local-variable name="a">
```

```
                    <type primitive="true" name="int"/>
                    <literal-number kind="integer" value="7"/>
                </local-variable>
                <local-variable name="b">
                    <type primitive="true" name="int"/>
                        <binary-expr op="-">
                        <binary-expr op="+">
                            <var-ref name="c"/>
                            <var-ref name="a"/>
                        </binary-expr>
                        <literal-number kind="integer" value="3"/>
                    </binary-expr>
                </local-variable>
            </block>
        </method>
    </class>
  </java-class-file>
</java-source-program>
```

intermedioXml.xml

Por último este es el aspecto de la instancias un vez generado el OWL. Nos centramos en la vista de los tipos declarados en el ejercicio, se puede ver en la figura 5.4 que se declaran tres enteros, un String y un void.

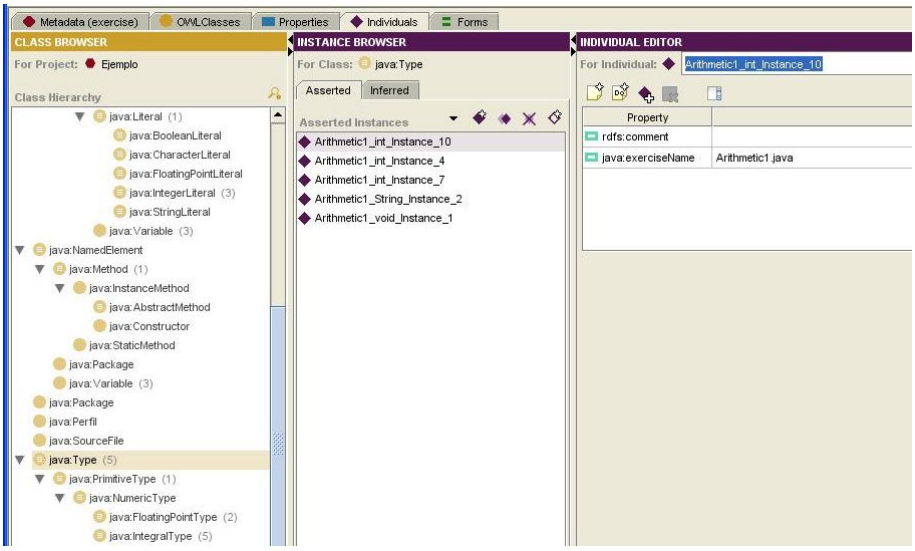

Figura 5.4: Aspecto de las instancias de la ontología con un ejercicio

Ejecución del ejercicio Java Arithmetic2 sobre Java2Owl

Para transformar este ejercicio a OWL los parámetros de entrada necesarios son los siguientes, téngase en cuenta que la salida se ha añadir al

archivo ya existente en esta ocasión y que queremos que se conserve el xml intermedio para poder mostrarlo:

Parámetros: -f Ejemplo.owl -k intermedioXml.xml Arithmetic2.java

Ahora la opción -n no debe incluirse para que la herramienta añada el contenido al archivo OWL ya existente.

```
class main {
        public static void main (String [] args ) {
                int c = 1;
                int a = 7;
                int b = c + (a - 3);
        }
} // main
```

Arithmetic2.java

El archivo xml intermedio es el siguiente:

```
<!-- Generated by Java2XML http://java2xml.dev.java.net/ -->
<java-source-program>
    <java-class-file name="Arithmetic2.java">
        <class name="main" visibility="protected">
            <method name="main" visibility="public" static="true">
                <type name="void" primitive="true"/>
                <formal-arguments>
                    <formal-argument name="args">
                        <type name="String" dimensions="1"/>
                    </formal-argument>
                </formal-arguments>
                <block>
                    <local-variable name="c">
                        <type primitive="true" name="int"/>
                        <literal-number kind="integer" value="1"/>
                    </local-variable>
                    <local-variable name="a">
                        <type primitive="true" name="int"/>
                        <literal-number kind="integer" value="7"/>
                    </local-variable>
                    <local-variable name="b">
                        <type primitive="true" name="int"/>
                        <binary-expr op="+">
                            <var-ref name="c"/>
                            <paren>
                                <binary-expr op="-">
                                    <var-ref name="a"/>
                                    <literal-number kind="integer" value="3"/>
                                </binary-expr>
                            </paren>
                        </binary-expr>
                    </local-variable>
                </block>
            </method>
        </class>
    </java-class-file>
```

```
</java-source-program>
```

intermedioXml.xml

Por último se puede ver en la figura 5.5 el aspecto de la instancias un vez generado el OWL. Nos centramos en la vista de las clases declaradas para ver como se ha añadido la clase main del ejercicio dos, sin borrar la instancia anterior del ejercicio 1.

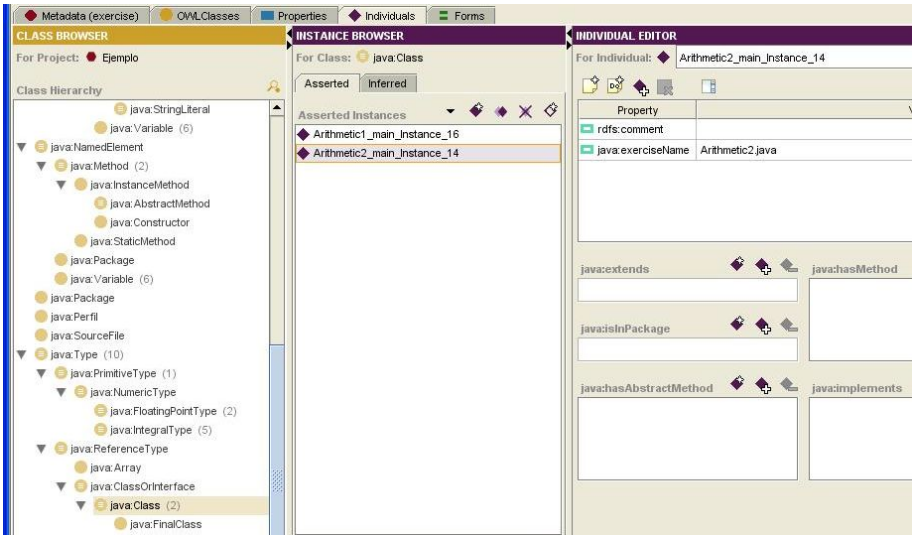

Figura 5.5: Aspecto de las instancias de la ontología con dos ejercicios

Ejecución del ejercicio Java Arithmetic3 sobre Java2Owl

Para transformar este ejercicio a OWL los parámetros de entrada necesarios son los siguientes, téngase en cuenta que la salida se ha añadir al archivo ya existente y que queremos que se conserve el xml intermedio para poder mostrarlo:

Parámetros: -f Ejemplo.owl -k intermedioXml.xml Arithmetic3.java

```
class main {
        public static void main (String [] args ) {
                double c = 1;
                int a = 7;
                int b = 3;
                b = b * a;
                c = b %( 2*a );
```

```
                c = c   / a;
        }
} // main
```

Arithmetic3.java

El archivo xml intermedio es el siguiente:

```
<!-- Generated by Java2XML http://java2xml.dev.java.net/ -->
<java-source-program>
    <java-class-file name="Arithmetic3.java">
        <class name="main" visibility="protected">
            <method name="main" visibility="public" static="true">
                <type name="void" primitive="true"/>
                <formal-arguments>
                    <formal-argument name="args">
                        <type name="String" dimensions="1"/>
                    </formal-argument>
                </formal-arguments>
                <block>
                    <local-variable name="c">
                        <type primitive="true" name="double"/>
                        <literal-number kind="integer" value="1"/>
                    </local-variable>
                    <local-variable name="a">
                        <type primitive="true" name="int"/>
                        <literal-number kind="integer" value="7"/>
                    </local-variable>
                    <local-variable name="b">
                        <type primitive="true" name="int"/>
                        <literal-number kind="integer" value="3"/>
                    </local-variable>
                    <assignment-expr op="=">
                        <lvalue>
                            <var-ref name="b"/>
                        </lvalue>
                        <binary-expr op="*">
                            <var-ref name="b"/>
                            <var-ref name="a"/>
                        </binary-expr>
                    </assignment-expr>
                    <assignment-expr op="=">
                        <lvalue>
                            <var-ref name="c"/>
                        </lvalue>
                        <binary-expr op="%">
                            <var-ref name="b"/>
                            <paren>
                                <binary-expr op="*">
                                    <literal-number kind="integer" value="2"/>
                                    <var-ref name="a"/>
                                </binary-expr>
                            </paren>
                        </binary-expr>
                    </assignment-expr>
                    <assignment-expr op="=">
                        <lvalue>
                            <var-ref name="c"/>
                        </lvalue>
                        <binary-expr op="/">
```

```
                                <var-ref name="c"/>
                                <var-ref name="a"/>
                            </binary-expr>
                        </assignment-expr>
                    </block>
                </method>
            </class>
        </java-class-file>
</java-source-program>
```

intermedioXml.xml

Por último se puede ver en la figura 5.6 el aspecto de la instancias un vez generado el OWL. Nos centramos en la vista de las variables existentes en la ontología una vez creado el archivo OWL sobre tres ejercicios.

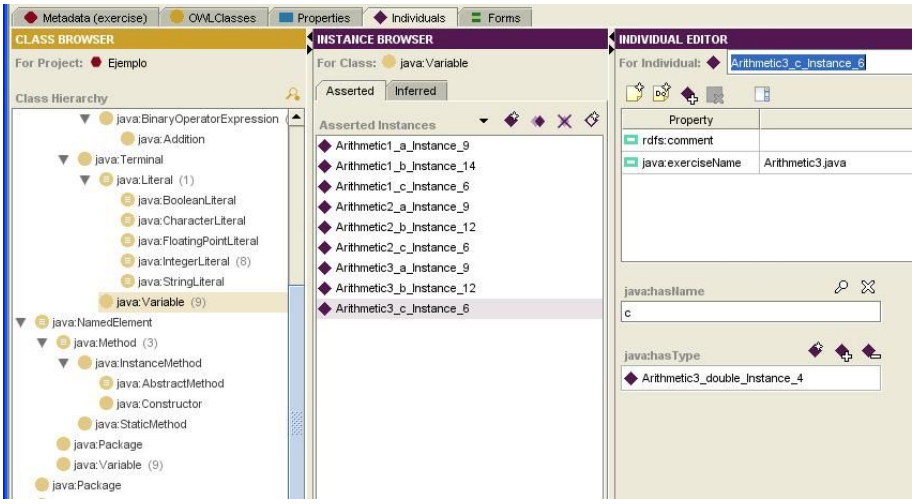

Figura 5.6: Aspecto de las instancias de la ontología con tres ejercicios

Para terminar este ejemplo de ejecución hemos usado la utilidad Jambalaya de Protegé (para información a cerca de Jambalaya consulte sección 3.4.1) para extraer una imagen de como quedan las instancias creadas dentro de la ontología, ya que una imagen global no era lo suficientemente clara hemos preferido mostrar el estado de las instancias creadas bajo el concepto *type*. En la imagen 5.7 se puede ver una instancia (cuadro morado) por cada tipo declarado en los ejercicios utilizados para el ejemplo, así como una línea saliendo de cada instancia hacia el exterior que representan las distintas relaciones que estas instancias pueden tener.

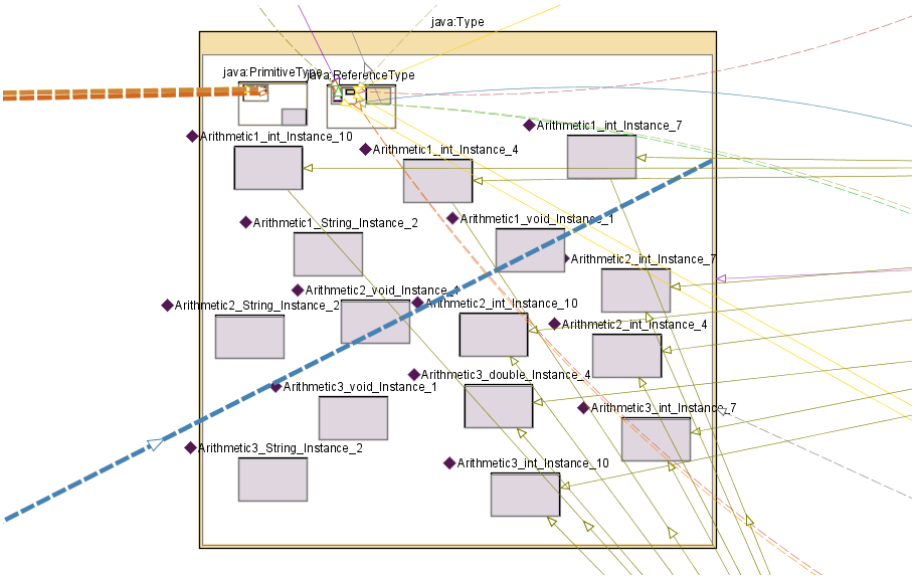

Figura 5.7: Grafo de instancias declaradas bajo el concepto Type

5.4.5. Trabajo Futuro

La herramienta Java2Owl, como hemos comentado a lo largo de todo el módulo, se encuentra con todo lo necesario para la conversión de ejercicios imperativos desarrollado, a falta de que la OntoJavy entre en la tercera versión de evolución, y los conceptos que todavía no habían sido añadidos pasen a estarlo.

De esta manera solo habría que observar detenidamente cuales son estos nuevos conceptos para ir al código de la herramienta y en la clase *Xml2owl* buscar el método parser correspondiente a cada nuevo concepto, y al final de cada método descomentar las partes que se indiquen mediante los comentarios y los ToDo.

Por ejemplo, si en la tercera versión de la Ontojavy incluyen al concepto **if** con todos sus descendientes, habría que buscar en la clase Xml2Owl el método *parseIfNode()* para descomentar la parte donde se instancia en la ontología el concepto, también habría que hacer lo mismo para los conceptos *trueCase*, *falseCase* y buscar en esta clase toda posible referencia que exista a tales métodos por si hubiera alguna llamada que estuviera comentada.

Una vez realizado esto habrá que ir a la clase *JavaOntology* y buscar los métodos *createNewIf()*, *createNewFalseCase()* y *createNewTrueCase()*,

y en cada uno de ellos comprobar la correspondencia entre el nombre de las constantes y el nombre de los conceptos así como el nombre de las propiedades que afecten a dichos conceptos. Si fuera necesario añadir código en estos métodos para manejar las propiedades que no se hayan tenido en cuenta hasta el momento. Una vez realizado todo esto habrá que probar su funcionamiento pasándole a la herramienta algún ejercicio donde se utilice un if, y observando la salida en formato OWL.

En el párrafo anterior hacemos una explicación detallada de como incluir a los conceptos que habiendo sido desarrollados por nosotros no estaban en la ontología. Igual de importante es explicar como añadir y donde el código necesario para tratar el resto de conceptos de la DTD de Java que todavía no ha sido desarrollado, este es el que concierne a los conceptos de ejercicios con orientación a objetos. Vamos a utilizar de nuevo un ejemplo real ya que nos parece más descriptivo. Si queremos añadir al código al concepto **constructor** habría que seguir los siguientes pasos:

1. Observar en la DTD al elemento constructor para ver cuantos y cuales son sus atributos y entidades dependientes. En este caso el elemento tiene el siguiente aspecto:

```
<!ELEMENT constructor (type-parameters?,formal-arguments,
        throws*,(super-call|this-call)?,(%stmt-elems;)?)>
<!ATTLIST constructor
    name CDATA #REQUIRED
    id ID #REQUIRED
    \%visibility-attribute;
    \%mod-final;
    \%mod-static;
    \%mod-synchronized;
    \%mod-volatile;
    \%mod-transient;
    \%mod-native;>
```

2. Crear el método *parseNewConstructorNode* en la clase *Xml2owl*. En el elemento de la DTD se puede observar que este es el encargado de invocar a los métodos parser correspondientes de *type-parameters* (en caso de que tenga), *formal-arguments*, *stmt-elems* (en caso de que tenga) , estos están ya implementados, y los todavía no implementados *throws*, *super-call* y *this-call*.

3. Observar los atributos de *constructor* para extraerlos mediante los métodos adecuados (todos ellos ya implementados), por ejemplo para el atributo “nameüsar *getNameAttr(Nodo)*.

4. Una vez que tenemos todos los atributos extraídos del nodo y hemos invocado a los métodos parser adecuados para continuar con el descenso en el árbol (implementando los que no estuvieran hechos), habrá que utilizar el atributo _ontology para crear las instancias y propiedades adecuadas en la ontología. Para ello invocamos a un método que llamaremos *createNewConstructor(atributos que consideremos necesarios)*. Este habrá que crearlo a su vez en la clase *JavaOntology*.

5. Ya en la clase *JavaOntology* habrá que crear el método antes citado, la constante correspondiente al concepto *constructor*, tal y como aparezca en la OntoJavy. También habrá que observar la OntoJavy para ver que propiedades conciernen a dicho concepto por si fuera necesario crear constantes nuevas para dichas propiedades.

6. Una vez realizado esto solo nos faltaría realizar el código del método *createNewConstructor(..)* correctamente para que se instancia tal concepto en la ontología con sus propiedades correspondientes según los atributos que tenga.

Este mismo proceso habrá que repetirlo con todos los conceptos de la DTD que no sean de tipo imperativo, a medida que la OntoJavy vaya evolucionando. Destacamos que la manera de tratar los espacios de nombres de las instancias ha de ser la misma que la que usamos nosotros y ya hemos explicado en este módulo.

5.5. Editor de perfil de usuario

Los perfiles de usuario serán en un futuro proporcionados por Javy2, que será el que acceda y modifique los mismos; pero en este momento esa parte no está aún implementada. Debido a esto, para poder probar nosotros el funcionamiento del sistema, necesitamos un interfaz de usuario que nos supla todas las funciones de Javy2 relacionadas con los perfiles[11].

Es por esto que utilizamos el editor de perfiles de usuario, desde el que un usuario externo podrá crear y modificar perfiles de forma fácil, añadiendo desde el principio los conceptos conocidos, a practicar y prohibidos, como se

[11] El editor de perfiles corresponde a una práctica realizada por Laura Gutiérrez Muñoz para la asignatura Ingeniería de Sistemas Basados en Conocimiento en el curso 2006-07.

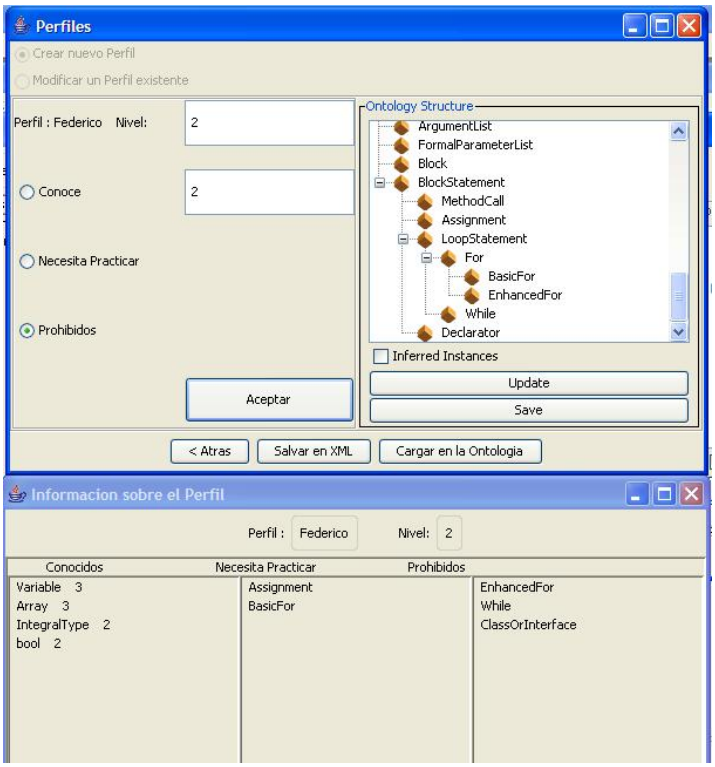

Figura 5.8: Captura de pantalla del editor de perfiles.

puede ver en la figura 5.5. Nótese que si en alguna categoría añadimos un nodo, se consideran implícitamente añadidos los nodos hijos.

Este interfaz tendrá las siguientes funciones:

- Creación de un nuevo perfil de usuario.
- Recuperación de un perfil existente. El interfaz enviará la orden al Manejador de perfiles para que éste recupere por el identificador el perfil de usuario desde el fichero XML, como se explicará en el apartado 5.6. Para cada perfil de usuario, el editor debe mostrar la siguiente información:
  - Lista de conceptos conocidos, junto con el nivel adquirido en cada uno de ellos. En el ejemplo, el perfil *Federico* conoce ya los conceptos *Variable*, *Array*, *IntegralType* y *bool* (el nivel adquirido en cada uno es, respectivamente, 3, 3, 2 y 2 cada concepto).
  - Lista de conceptos prohibidos. *Federico* todavía no está preparado para *EnhancedFor*, *While* ni ningún tipo de *ClassOrInterface*.

  - Lista de conceptos que necesita practicar. En este ejemplo, necesita practicar los conceptos *Assignment* y *BasicFor*.
  - Nivel general del usuario (número de ejercicios completos resueltos). En el ejemplo, *Federico* ha resuelto 2 ejercicios.

- Una vez recuperado, el usuario podrá modificar el perfil, realizando las siguientes acciones:

  - Añadir conceptos a cualquiera de las listas. Para ello se mostrarán grafos extraídos de la ontología sobre conceptos de compilación, de los que el usuario elegirá los nodos cuyos nombres se correspondan con el concepto deseado.
  - Eliminar conceptos de las listas de conceptos prohibidos y conceptos para practicar.
  - Modificar el nivel de usuario.
  - Guardar cambios. Entonces debe enviarse al **Manejador de perfiles de usuario** la orden de modificación, que modificará el perfil en el fichero XML,como se explicará en el apartado 5.6.
  - Si el perfil se encuentra como usuario actual de la aplicación, podrá dejar de serlo. En ese caso el editor mandará la información necesaria al **Manejador de perfiles de usuario**, que se encargará de borrar de la ontología el individuo correspondiente a dicho perfil, como se explicará en el apartado 5.6.
  - Si por el contrario el perfil no es el actual de la aplicación, existirá la opción de ponerlo como tal. Si existe otro perfil como actual, el editor lo notificará, y si el usuario así lo desea, se enviará al Manejador la orden de eliminar de la ontología el perfil que se encontraba como usuario actual, e introducir éste como tal.

Aunque el editor permite guardar perfiles de usuario como código OWL, nosotros no utilizamos esta funcionalidad, abarcada por el recomendador de ejercicios. La mayor ventaja de esta herramienta es la de poder tratar con los perfiles de usuario de una manera fácil y cómoda, sin necesitad de conocer de memoria los nombres de los conceptos de compilación, simplemente marcándolos y agregándolos, pudiendo después insertarlos y recuperarlos de la base de datos XML sólo con un botón, siendo el editor el encargado de la comunicación con el Manejador de escenarios.

5.6. Manejador de perfiles de usuario:

El manejador de perfiles de usuario es el módulo encargado de gestionar los perfiles de usuario almacenados en XML. Recibe las órdenes del **Editor de perfiles de usuario**. Por orden suya, este manejador podrá insertar, recuperar y modificar perfiles en el fichero XML.

5.6.1. Descripción del perfil de usuario

El perfil de usuario se basa en varios factores:

1. **Nombre de usuario**: Nombre único que identifica a cada jugador.
2. **Conceptos aprendidos hasta el momento**: Indica cuántas veces ha sido resuelta la parte de algún ejercicio que complete un concepto de compilación.
3. **Conceptos aprendidos hasta el momento**: Indica cuántas veces ha sido resuelta la parte de algún ejercicio que complete un concepto de compilación.
4. **Conceptos a practicar**: Siguientes conceptos que el usuario está preparado para aprender.
5. **Conceptos prohibidos**: Conceptos para los que el usuario bajo ningún concepto está preparado para aprender.
6. **Diferentes factores referentes a la destreza del jugador** en el entorno del juego: rapidez de resolución, número de porta-recursos capturados...(aún por determinar).

En un futuro, el nombre de usuario y los ejercicios practicados serán proporcionados por Javy2, y será un agente pedagógico el que determine los conceptos que el usuario debe practicar y los que le son prohibidos[12].

5.6.2. Organización

El manejador de perfiles a su vez se divide en dos módulos o clases diferentes (figura 5.9):

[12]En nuestro caso, esta información nos viene dada por el Editor de Perfiles, por lo que los aspectos pedagógicos de la información contenida en el perfil no serán tenidos en cuenta en nuestro proyecto

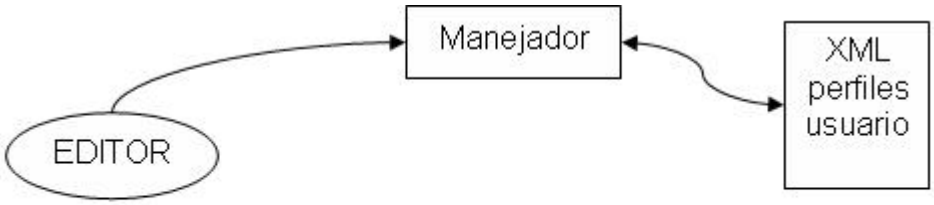

Figura 5.9: Estructura general del manejador de perfiles.

- **Manejador**: El manejador es el módulo que canaliza las órdenes provenientes del **Editor de perfiles de usuario** hacia PerfilesXML , de forma que por sí mismo no se relaciona con XML.
- **Módulo PerfilesXML**: El usuario introduce o modifica perfiles de usuario a través del editor. Éstos se crearán o modificarán en el XML por el módulo Perfiles XML.

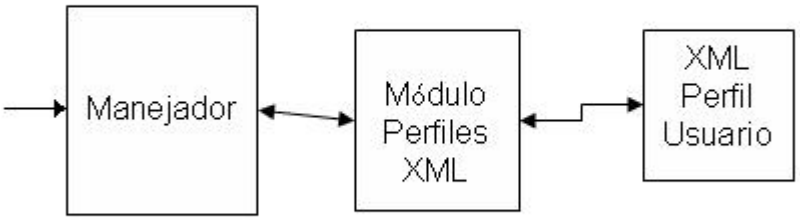

Figura 5.10: Relaciones entre el manejador de perfiles.

5.7. Recomendador CBR

En este apartado hablaremos del módulo principal, encargado de encontrar el ejercicio más adecuado de entre todos los posibles para cada usuario, basándose en el perfil de usuario.

El Recomendador por medio de una interfaz gráfica solicita el id o nombre del perfil asociado al usuario al que se quiere recomendar el ejercicio. Una vez obtenido el nombre, le pide al Manejador de perfiles el Perfil de Usuario asociado al mismo. El manejador le devuelve el perfil, y es entonces cuando un sub-módulo del recomendador introduce el perfil en la ontología sobre conceptos de compilación en la que se encuentra la base de ejercicios en código OWL (consultar sección 5.7.1).

Cuando el perfil ya esté insertado, otro sub-módulo se encarga de elegir de entre los ejercicios de la ontología aquellos que cumplan con el algoritmo de recuperación representacional escogido. De entre estos ejercicios, con funciones de similitud computacionales se consigue una lista de ejercicios propuestos, ordenados desde los que se encuentren más a menos similares al ejercicio ideal para el usuario.

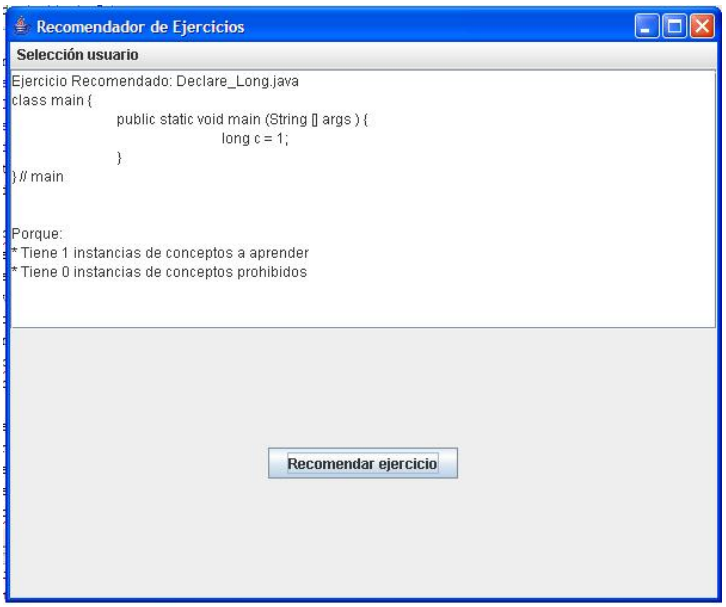

Figura 5.11: Interfaz gráfico del recomendador de ejercicios con una de las soluciones mostradas para un perfil

Una vez elegido el ejercicio para recomendar, el interfaz muestra por pantalla su nombre, el código java del ejercicio y el porqué de la elección (ver figura 5.11). Además, al pulsar el botón de “Recomendar ejercicio”se recomienda un nuevo ejercicio, que puede no ser el mismo si existen varios ejercicios igualmente recomendables.

5.7.1. Inserción del perfil de usuario en OWL y Funciones de similitud

Al llegar al momento de decidir la función de similitud y los algoritmos de recuperación se nos abrieron muchas posibilidades. Por un lado teníamos dos opciones de recuperación, **computacional** (Mediante reglas y cómputos), o **representacional** (Con recuperaciones en la ontología en función del parentesco con el perfil).

La recuperación representacional nos parecía importante para aprovechar las facilidades que nos ofrecía el hecho de tener el perfil de usuario en la ontología, pero no creímos que fuese suficientemente concreto. Por otro lado, la computacional nos parecía más intuitiva y concreta, con más opciones de funciones de similitud. Sin embargo, con ella desaprovechamos el hecho de tener el perfil en la ontología (tendríamos que sacar todos los ejercicios de la ontología para realizar el cómputo); y además el cómputo sería demasiado complicado por contar con una gran base de ejercicios.

Por estas razones decidimos hacer una mezcla de los dos tipos, primero haríamos una recuperación desde la ontología que nos sirviera de criba para luego, con una base de ejercicios más reducida realizar la función de similitud por algún método computacional. Aún así teníamos diferentes posibilidades, así que nos dedicamos a estudiarlas, con el fin de quedarnos con aquella que nos pareciera que da mejor resultado, desde el punto de vista de acierto y de coste.

Primera opción: familia de los elementos practicados.

Consiste en introducir en la Ontología de Conceptos de Compilación los datos del perfil de usuario correspondientes a los ejercicios practicados. Estos nodos se introducen como individuos de los conceptos corespondientes a los de los ejercicios, y de ahí mediante "relaciones de familia" (es decir, cogiendo los individuos hermanos, primos, etc) escogemos un grupo de ejercicios candidatos. Pero ya aquí teníamos que decidir cuáles serían las relaciones de parentesco que nos interesaban. Por ejemplo, suponemos que nuestro perfil de usuario tenía un concepto conocido de tipo **java:PrimitiveType** (ver figura 5.12). En este caso, ¿qué ejercicios nos interesarían?. Entonces pensamos si nos deberíamos quedar con ejercicios que tuvieran conceptos clasificados en la ontología como "hijos" de java:PrimitiveType. En este caso nuestra decisión fue inmediata: si el usuario ha practicado el concepto

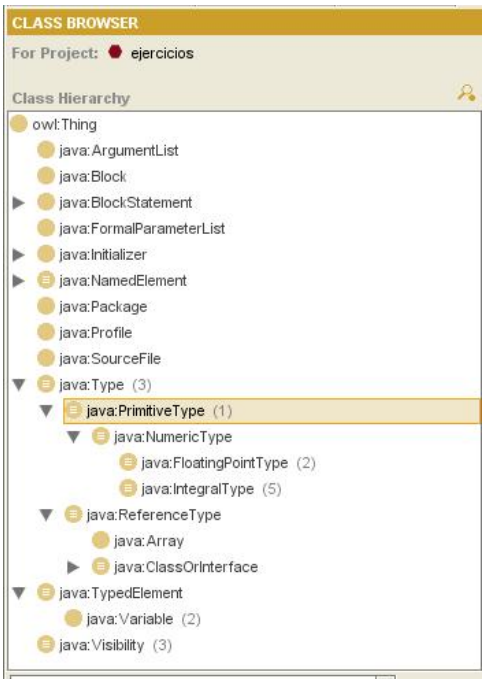

Figura 5.12: Ejemplo 1: conceptos practicados

java:PrimitiveType, es porque habrá resuelto ejercicios con los conceptos hijos; así que decidimos coger también los ejercicios asociados a los nodos "hijo". De esta forma, además, conseguimos una lista más amplia de candidatos.

Pero esta no era la única decisión, ya que debemos tener en cuenta que el usuario ya ha practicado este tipo de concepto (en este caso java:PrimitiveType), por lo que no podemos seleccionar sólo con los hermanos (no se haría más que repetir una y otra vez el mismo tipo de ejercicio). Así que ya nos quedó claro que debíamos mirar más allá. Así que decidimos coger también los conceptos primos. Ahora la recuperación devolvería también los ejercicios con conceptos de tipo **java:ReferenceType**. En este ejemplo, creemos acertado que una vez que el usuario conozca los tipos primitivos, comience con los referenciados. Pero aquí nos encontramos otra vez con la duda: ¿también cogemos los ejercicios asociados a los nodos hijos? La respuesta, por las mismas razones que anteriormente, fue positiva (para conocer los conceptos java:ReferenceType hay que conocer todos sus hijos). Ahora la base de ejercicios recuperados se hacía muy grande, y además incluye los ejercicios con conceptos asociados a clases e interfaces (**java:ClassOrInterface**); y eso teniendo en cuenta que cada usuario suele tener varios conceptos conocidos. Luego la función de similitud computacional eliminaría muchos de estos con-

ceptos, pero en muchos casos se podría llegar a devolver de la recuperación representacional toda la base de ejercicios inicial; por lo que ésta no nos serviría de mucha ayuda.

Decidimos, por tanto, desechar esta opción.

Segunda opción: familia de los elementos que el usuario necesita practicar.

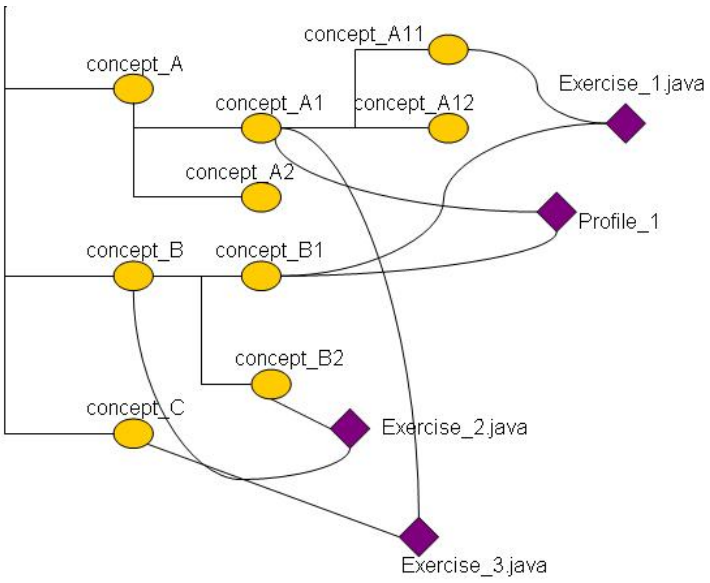

Figura 5.13: Ejemplo 2: Perfil de Usuario introducidos en la ontología por sus conceptos a aprender.

Tras ver que basar la recuperación representacional en los conceptos practicados por el usuario nos devolvía una cantidad de ejercicios candidatos demasiado abundante, como siguiente paso pensamos en incluir en la ontología, en vez de los ejercicios practicados, los ejercicios a practicar. Así, además, aprovechamos esta información, buscando ya directamente ejercicios con conceptos que se deben practicar, evitando los ejercicios sin más conceptos que los que el usuario conoce ya suficientemente, lo que provoca el hastío por falta de metas.

Supongamos, por ejemplo, la figura 5.13[13].

[13]Es importante señalar que los ejercicios no están incluidos en la ontología como se

En este ejemplo, el usuario asociado al individuo *Profile_1* necesita practicar los conceptos *Concept_A1* y *Concept_B1*. Por lo tanto, los ejercicios que nos interesa seleccionar son, en principio, *Exercise_1.java* y *Exercise_3.java*; es decir, los "hermanos" de *Profile_1*. Pero entonces nos planteamos si sería adecuado escoger los ejercicios "primos" de *Profile_1*.

Esta posibilidad fue desestimada, al tener en cuenta que si un usuario necesita practicar un determinado concepto, esto no implica que esté preparado para otro parecido. Esto ocurre en muchas facetas de la vida, por ejemplo, en matemáticas. Suponemos un chico de primeros cursos del colegio al que se le va a enseñar a sumar. Estará en buena situación para practicar sumas (de hecho, ese es el método para que aprenda), pero a ningún profesor se le ocurriría ponerle hacer restas hasta que las sumas no estuvieran más o menos dominadas, aunque las dos operaciones son, para nosotros, muy parecidas.

Así que decidimos tomar de la ontología solo los ejercicios *Concept_A1* y *Concept_B1*. Pero ahora, ¿cómo seleccionamos entre los 2 cuál es mejor? Para esto parecía adecuada en principio una función de similitud computacional.

Nuestra primera idea, dado que los ejercicios propuestos inicialmente desde la ontología pueden tener conceptos prohibidos, era eliminar los que los tuvieran. Así que lo que necesitábamos saber era los ejercicios con conceptos, de entre los propuestos, tenían ejercicios prohibidos. La única forma de obtenerlos era otra vez a través de la ontología, así que la función no iba a ser para este caso computacional, sino relacional. A raíz de esta idea nos planteamos una tercera opción.

Tercera opción: familia de los elementos que el usuario necesita practicar menos los prohibidos.

Ya que íbamos a tener que buscar dos veces en la ontología (una vez los ejercicios con conceptos a practicar y otra los que contuvieran conceptos prohibidos), decidimos hacerlo en una primera fase. Así, para cada perfil, decidimos añadir un individuo en la ontología que fuera instancia de un

representa en la imagen, pero hemos preferido mostrar este esquema para mayor claridad en la explicación. El formato de inserción de los ejercicios se puede consultar en ella subsección 5.4.3

concepto nuevo, *Profile*, y que a su vez se uniera a otros nodos de la ontología mediante dos nuevas relaciones:

- **forbidden**: Señala los conceptos que el perfil tiene como prohibidos.
- **has_to_practice**: Relación con los conceptos que el usuario debería practicar.

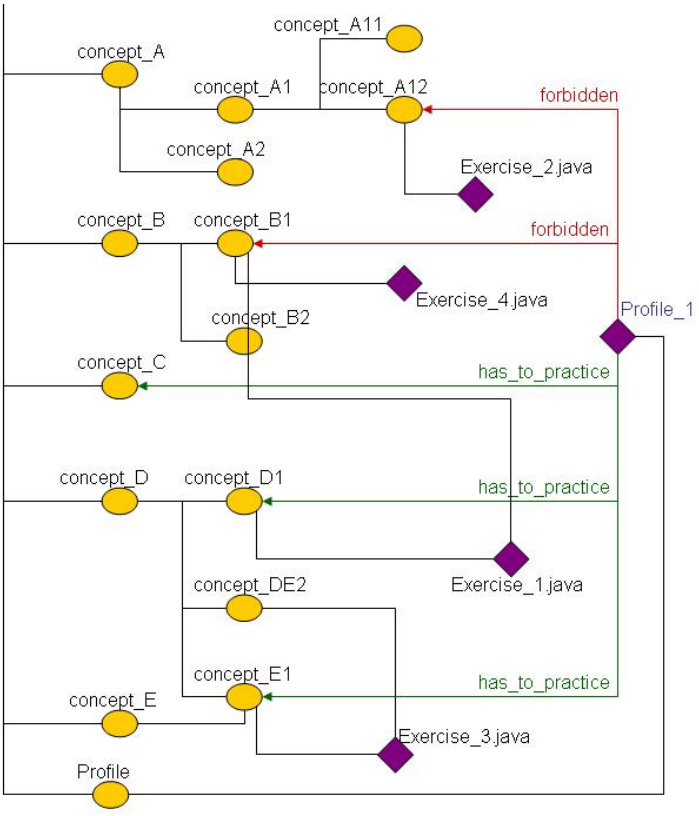

Figura 5.14: Ejemplo 3: Perfil de Usuario introducidos con relaciones de sus conceptos a aprender y aprendidos.

Tomamos como ejemplo la figura 5.14. En ella, el usuario *Profile_1* tiene como conceptos a aprender *Concept_D1* y *Concept_E1* (con las líneas de relación en verde); y como conceptos prohibidos *Concept_A12* y *Concept_B1* (con líneas de relación rojas).

En una primera pasada, se elegirán aquellos ejercicios con conceptos que el usuario debe practicar[14]. Por ello se elegirán *Exercise_1.java* y *Exerci-*

[14]Cabe señalar que lo que extraemos de la ontología no son los ejercicios Java en sí, sino una seña identificativa del mismo, tal como su nombre

se_3.java. Por otro lado, se seleccionan los ejercicios con conceptos que el perfil de usuario marca como prohibidos. Estos son *Exercise_1.java*, *Exercise_2.java* y *Exercise_4.java*. Una vez obtenidos los dos conjuntos, hacemos una diferencia de conjuntos para obtener los ejercicios candidatos:

$$EjerciciosCandidatos = EjerciciosRecomendados \setminus EjerciciosProhibidos$$

Consideramos EjerciciosRecomendados el conjunto de ejercicios con conceptos a practicar, EjerciciosProhibidos el conjunto de ejercicios con conceptos prohibidos y EjerciciosCandidatos el conjunto que ha superado la primera fase de selección.

Esta operación, en nuestro ejemplo, será:

$EjerciciosCandidatos = \{Exercise\_1.java, Exercise\_3.java\ \} \setminus \{Exercise\_2.java,\ Exercise\_4.java, Exercise\_1.java\}$

$EjerciciosCandidatos = \{Exercise\_3.java\}$

Así, el ejercicio elegido será *Exercise_3.java*.

En este caso no nos ha hecho falta más que la recuperación representacional, pero nos podemos encontrar con que la diferencia de conjuntos nos proporcionara más de un ejercicio candidato, o que, por el contrario, no nos devolviera ningún ejercicio. Nos faltaban pues varias decisiones que tomar.

En los siguientes párrafos describiremos las distintas evaluaciones de similitud posibles que fuimos considerando hasta llegar a la actual.

Recuperación computacional de los ejercicios.

Para decidir las funciones de similitud a obtener tomamos en cuenta dos posibles conjuntos de ejercicios candidatos: aquellos con varios ejercicios y conjuntos vacíos. El grupo de ejercicios candidatos será vacío cuando no exista en la ontología ningún ejercicio con conceptos a practicar pero sin

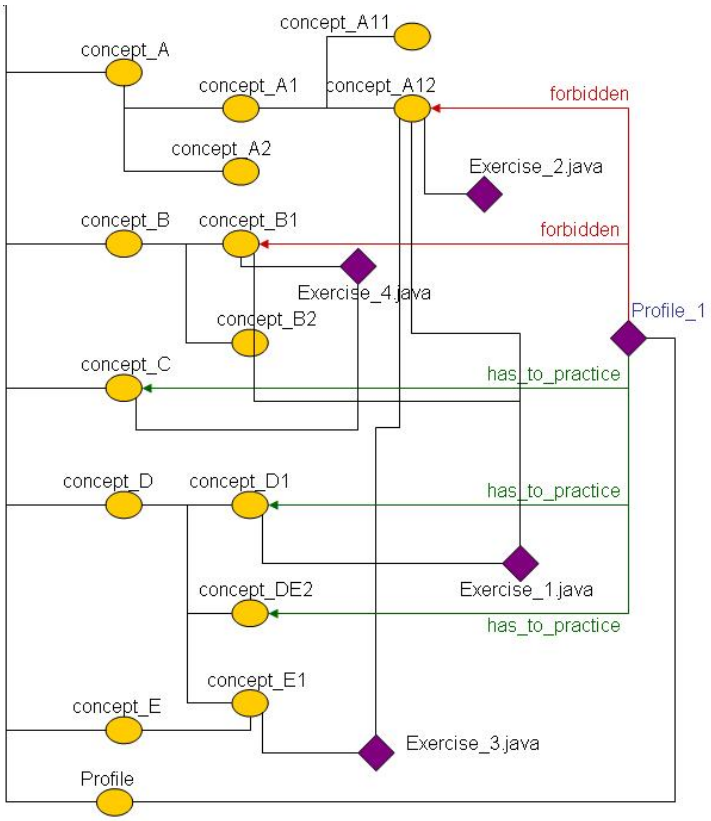

Figura 5.15: Ejemplo 4: Perfil de Usuario introducidos con relaciones de sus conceptos a aprender y aprendidos.

conceptos prohibidos.

Téngase en cuenta, por ejemplo, la figura 5.15. En este caso, el ejercicio *Exercise_1.java* tiene dos conceptos prohibidos para el perfil, y el resto uno. Los ejercicios que pertenecerán al conjunto "EjerciciosRecomendados" serían *Exercise_1.java* y *Exercise_4.java* ; y los pertenecientes al conjunto EjerciciosProhibidos, *Exercise_1.java*, *Exercise_2.java*, *Exercise_3.java* y *Exercise_4.java*. Por lo tanto, la diferencia de conjuntos es

$$EjerciciosCandidatos = \{Exercise\_1.java, Exercise\_4.java\ \} \setminus \{Exercise\_1.java,\ Exercise\_2.java, Exercise\_3.java, Exercise\_4.java\}$$

$$EjerciciosCandidatos = \ \phi$$

Ante esta situación (que con una buena cantidad de ejercicios no debería ser habitual, pero debemos tomarla en cuenta), decidimos elegir aquél ejercicio con menos ejercicios prohibidos, por la razón de que puesto a hacer un ejercicio con conceptos que no debería hacer, cuantos menos tenga mejor, y creemos más favorable esto que indicar al usuario que no quedan ejercicios para él. Así, el conjunto de nuevos candidatos estará compuesto por el ejercicio *Exercise_4.java*.

En el caso, mucho más probable, de que haya más de un ejercicio entre los candidatos, teníamos que decidir qué criterio seguir para ordenar el conjunto según su preferencia. En este caso nos planteamos cuál era la mejor forma de que un alumno aprendiera: resolviendo ejercicios con el mayor número de ejercicios a practicar (lo que conlleva una mayor dificultad) o poco a poco, afianzando los conocimientos y practicando pocos conceptos a practicar cada vez. Nuestra elección estaba en realidad en competencia del módulo pedagógico mencionado anteriormente[15], pero al no estar éste disponible, tras muchas discusiones nos decantamos por esta segunda opción, por no complicar demasiado cada nivel, además de para contar con más ejercicios que ofrecer, porque si en un primer momento recomendamos aquél con más conceptos a practicar, eliminaríamos en cierto modo todos los demás[16].

Y en el caso de que haya más de un ejercicio con el mismo número mínimo de conceptos a practicar, la elección decidimos que fuese aleatoria.

5.7.2. Desarrollo

El recomendador de ejercicios debe ser capaz de proponer un ejercicio a un usuario o jugador de Javy2. El recomendador obtendrá una serie de conceptos que el jugador ya conocerá, deberá a aprender, o que no debe resolver todavía. Estos conceptos serán proporcionados por un módulo pedagógico ajeno a éste. Con dichos conceptos este módulo será capaz de proporcionar el ejercicio más adecuado para el aprendizaje del usuario.

[15]Este módulo es un proyecto futuro del departamento de GAIA de la FDI para integrarlo en Javy2

[16]Estos ejercicios se ofrecerían seguramente más adelante, pero eso podría provocar ciertas incongruencias, ya que una vez que el usuario llegase a un "llano" en sus nuevos conocimientos se ofrecerían ejercicios cada vez más fáciles.

Estructura General

El recomendador debe completar una serie de tareas para devolver el ejercicio más adecuado para un jugador, como se muestra gráficamente en la figura 5.16.

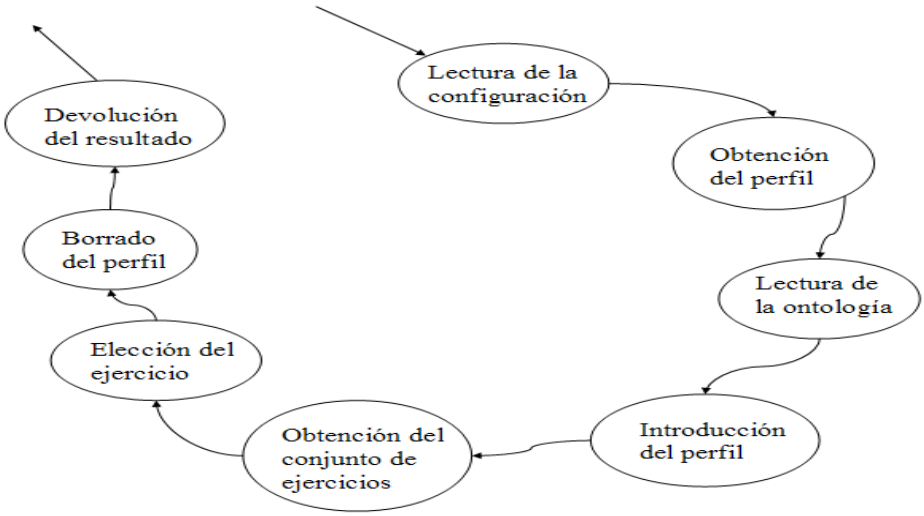

Figura 5.16: Tareas que debe completar el recomendador de ejercicios

La ejecución de estos pasos está controlada por una única clase, que sirve de punto de acceso, llamada Recomendador. Lo que tiene que realizar en cada paso es:

1. **Lectura de la configuración:** En este paso el sistema lee una serie de parámetros necesarios para su funcionamiento, como por ejemplo el nombre del usuario al que se le va a recomendar el ejercicio o la ruta de la ontología donde están los ejercicios.
2. **Obtención del perfil:** Con el nombre del usuario al que se le va a recomendar el ejercicio en este paso el Recomendador requerirá al módulo PerfilAlumno que le proporcione todos los datos del perfil de ese alumno. Entre esos datos estarán la lista de conceptos aprendidos, la lista de conceptos a aprender, y la lista de conceptos prohibidos.
3. **Lectura de la ontología:** En esta tarea se leerá la ontología con todos los ejercicios de la base de casos.
4. **Introducción del perfil:** En esta tarea se introducirá en la ontología anteriormente leída el perfil del alumno, incluyendo su nombre, lista de conceptos conocidos, a aprender, y prohibidos.

5. **Obtención del conjunto de ejercicios:** En este paso se obtendrá un conjunto de ejercicios que son adecuados para proponer al usuario.
6. **Elección del ejercicio:** De entre todos los ejercicios que son adecuados se seleccionará uno, que es el que finalmente se propondrá al usuario.
7. **Borrado del perfil:** Tras extraer el ejercicio propuesto se borrará el perfil de la ontología así como todas los conceptos a aprender, conocidos o prohibidos de ese usuario.
8. **Devolución del resultado:** Se devolverá el ejercicio propuesto al usuario.

Las siguientes secciones detallan cada uno de estos pasos.

Lectura de la configuración

En esta fase el recomendador analizará el fichero xml situado en `/scripts/datos_recomendador/configRecomendador.xml` extrayendo los parámetros necesarios para su funcionamiento. Los parámetros que lee del fichero xml son:

- La ruta del archivo donde se almacenan los perfiles de los usuarios.
- El nombre del usuario al que se le va a recomendar el ejercicio. Con este nombre y conociendo el archivo donde se almacenan los perfiles podremos obtener el perfil de un usuario concreto.
- La ruta de la ontología que define todos los conceptos de Java.
- La ruta del archivo donde están almacenados todos los ejercicios de la base de casos. Este archivo contiene una serie de instancias de los conceptos definidos en la ontología sobre Java. Cada instancia se corresponde con la aparición de un concepto de la ontología sobre Java en los ejercicios. Esto es, si en un ejercicio aparece una variable x, en este archivo habrá una instancia del concepto variable de nombre x y que tendrá una propiedad con el nombre del ejercicio en el que aparece.
- La URI del ejercicio. Identificador que asociará a la ontología, necesario para la creación de la ontología en memoria al leer del fichero con las instancias de los conceptos.
- El directorio donde se sitúan los ejercicios de la base de casos en formato java. Éste es necesario para mostrar el ejercicio finalmente propuesto.

Obtención del perfil

En esta fase se crea una instancia del módulo PerfilAlumno, ya descrito en la sección 5.6.1, indicándole la ruta del fichero con todos los perfiles de los usuarios. Tras su creación, se le indica al módulo PerfilAlumno que proporcione el perfil del usuario al que se le va a recomendar un ejercicio. Este perfil contiene información con los conceptos de la ontología sobre Java que el usuario ya conoce, los que debe aprender o practicar, y los que tiene prohibidos, es decir, que todavía no debe practicar. Esta información habrá sido actualizada en el archivo de perfiles por un módulo pedagógico independiente de este módulo.

Lectura de la ontología

En esta fase se lee la ontología con los ejercicios de la base de casos como instancias de conceptos de la ontología sobre Java. Para la lectura de este archivo de instancias es necesario leer primero la ontología sobre Java y posteriormente leer el archivo con las instancias diciendo que sigue esa ontología. Finalmente se obtendrá una ontología con múltiples instancias. Cada instancia representará una aparición de un concepto de la ontología sobre Java en los ejercicios. Por ejemplo, si un ejercicio tiene una variable x, se creará una instancia del concepto variable de nombre x con la propiedad de que aparece en ese ejercicio. Si otro ejercicio posee también una variable de nombre x, se creará una nueva instancia de nombre x con la propiedad que indique que una variable x aparece en este nuevo ejercicio.

El archivo con las instancias de los ejercicios habrá sido creado con la herramienta descrita en la sección 5.4, aplicándola sucesivamente a todos los ejercicios de la base de casos.

Introducción del perfil en la ontología

En esta fase se introduce el perfil del alumno en la ontología obtenida. Primero se introduce una instancia del concepto perfil con el nombre del usuario, posteriormente para cada concepto de la lista de conceptos aprendidos, a practicar, y que no debe practicar se crea una instancia con un nombre genérico, para poder identificarlo, y sin estar asociado a ningún ejercicio.

Para los conceptos de la lista de conceptos aprendidos se introduce una instancia del concepto que referencian de nombre *known_concept@nombre_del_concepto*,

por ejemplo, si el concepto variable estuviese en la lista de conceptos aprendidos, se crearía una instancia del concepto variable de nombre "known_concept@variable"sin estar asociado a ningún ejercicio.

Para los conceptos de la lista de conceptos a practicar se introduce una instancia del concepto que referencian de nombre *practice_concept@nombre_del_concepto*, por ejemplo si el concepto if estuviese en la lista de conceptos a practicar se crearía una instancia del concepto if de nombre "practice_concept@if" sin estar asociado a ningún ejercicio.

Para los conceptos de la lista de conceptos prohibidos se introduce una instancia del concepto que referencian de nombre *forbidden_concept@nombre_del_concepto*, por ejemplo si el concepto while estuviese en la lista de conceptos prohibidos se crearía una instancia del concepto while de nombre "forbidden_concept@while" sin estar asociado a ningún ejercicio.

Estas instancias de conceptos de la ontología sin ningún ejercicio asociado son las que llamamos instancias de conceptos genéricas. Cada instancia genérica estará asociada con la instancia del concepto perfil del usuario actual mediante una propiedad que depende del tipo de concepto con el que tratamos, aprendido, a practicar, o prohibido.

Los conceptos en la lista de conceptos aprendidos se relacionan con el perfil mediante la propiedad *hasKnowledge*, los conceptos en la lista de conceptos a practicar se relacionan con el perfil mediante la propiedad *hasToPractice*, y los conceptos en la lista de conceptos prohibidos se relacionan con el perfil mediante la propiedad *mustNotPractice*. Estas propiedades se crean tras crear la instancias de conceptos genéricos.

Podemos ver un ejemplo en la figura 5.17, en esta figura se observa la instancia del concepto perfil, en este ejemplo el usuario se llama "Elena". Esta instancia se relaciona con las instancias de los conceptos genéricas mediante las propiedades *hasKnowledge*, representada por una flecha verde, *hasToPractice*, representada por una flecha azul, y *mustNotPractice*, representada por una flecha roja. Las instancias de los conceptos genéricas aparecen en diferente color según sean de la lista de conceptos aprendidos, en verde, de la lista de conceptos a aprender, en azul, o de la lista de conceptos prohibidos, en rojo. En este ejemplo la alumna "Elena" ya conoce el concepto Variable, debe practicar el concepto If, y todavía no debe practicar los conceptos For o While.

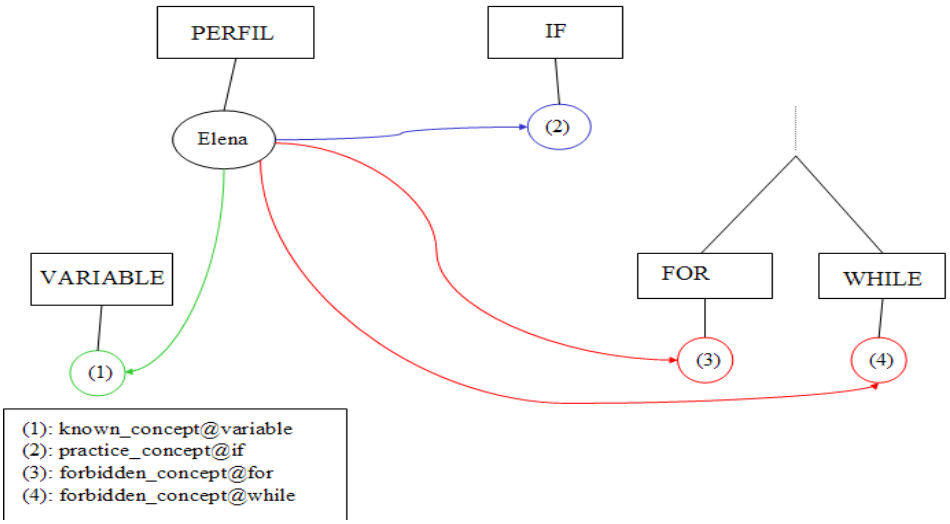

Figura 5.17: Ejemplo de un perfil introducido en la ontología

Obtención del conjunto de ejercicios propuestos

En esta fase se selecciona el conjunto de ejercicios que serán aptos para el alumno. Actualmente se considera que un ejercicio es apto para que el alumno lo realice si posee instancias de conceptos que debe practicar y no posee instancias de conceptos que no debe practicar. Esta es solo una opción de las que se podrían aprender, si el objetivo del sistema fuese reforzar los conocimientos adquiridos en vez de aprender nuevos podría considerarse un ejercicio apto si tuviese instancias de conceptos conocidos y ninguna de conceptos prohibidos o a aprender. Esta es una elección que corresponde al módulo pedagógico y con la estrategia actual si se quisiese reforzar los conocimientos del alumno se podrían incluir dichos conocimientos en la lista de conceptos a aprender.

Para obtener el conjunto de ejercicios recomendables primero extraemos los ejercicios con instancias de conceptos a aprender. Esto lo hacemos recorriendo la lista de conceptos a aprender, cada una de las propiedades *hasToPractice*, recorriendo cada uno de los hermanos de la instancia de ese concepto genérica, es decir, los hijos del concepto que aparecen en algún ejercicio, y extrayendo los nombres de los ejercicios. Estos nombres de ejercicios se almacenan en una lista sin repeticiones pero contando el número de instancias que tiene cada uno. Por ejemplo, si el ejercicio A tuviese 2 instancias de conceptos a aprender y el ejercicio B tuviese 7 instancias de conceptos a aprender, esa lista estaría formada por dos elementos, el primero el ejercicio A, con un atributo que indicase que tiene dos instancias

de conceptos a aprender, y el ejercicio B, con un atributo que indicase que tiene 7 instancias de conceptos a aprender.

Tras obtener el conjunto de ejercicios con instancias de conceptos a aprender se procede de igual forma para obtener el conjunto de ejercicios con instancias de conceptos prohibidos, obteniéndose finalmente dos conjuntos, uno con los ejercicios con instancias de conceptos a aprender y el número de instancias en cada ejercicio, y otro con los ejercicios con instancias de conceptos prohibidos y el número de instancias en cada ejercicio.

Para obtener el conjunto de ejercicios aptos se resta el conjunto de ejercicios con instancias de conceptos a aprender menos el conjunto de ejercicios con instancias de conceptos prohibidos. Si este conjunto resultase vacío se propondría como conjunto de ejercicios aptos el conjunto de ejercicios con instancias de conceptos a aprender, ya veremos como seleccionaremos el ejercicio propuesto de este conjunto en la sección 5.7.2.

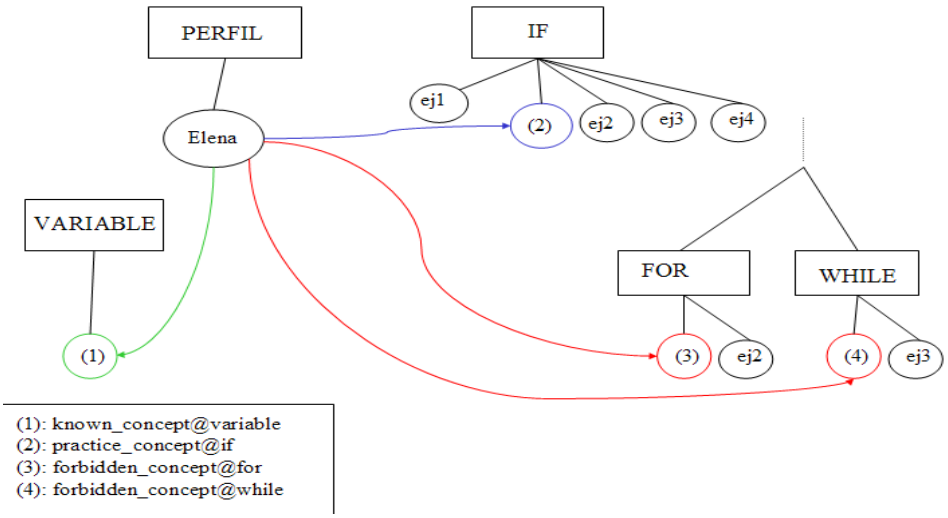

Figura 5.18: Ejemplo de elección del conjunto de ejercicios recomendables.

En el ejemplo de la figura 5.18 se ha introducido el perfil del usuario "Elena" con los conceptos conocidos, a practicar, y prohibidos, como ya vimos en la figura 5.17. En este ejemplo vemos además las instancias de conceptos de la ontología con un ejercicio asociado, y, aunque en la figura se muestran como si el nombre de la instancia fuese el nombre del ejercicio, realmente el nombre de la instancia es otro, y esta instancia tiene asociado, mediante una propiedad, el nombre del ejercicio, aunque en la figura se muestra

así por mayor claridad. Por ejemplo el ejercicio "ej1" tiene una instancia del concepto a aprender "If".

En el ejemplo el conjunto de ejercicios con instancias de conceptos a aprender sería el {ej1,ej2,ej3,ej4} y el conjunto de ejercicios con instancias de conceptos prohibidos sería {ej2,ej3}, de forma que finalmente el conjunto de ejercicios recomendable sería el {ej1,ej4}.

Elección del ejercicio propuesto

Una vez obtenido el conjunto de ejercicios recomendable esta fase se encarga de elegir uno para proponer al alumno. Aquí hemos de distinguir dos casos:

1. La resta entre los conjuntos de ejercicios con instancias de conceptos a aprender menos el conjunto de ejercicios con instancias de conceptos prohibidos no ha resultado vacía: En este caso los ejercicios se ordenan de menor a mayor según el número de instancias de conceptos a aprender que posean. Esto es así pues se ha considerado que el alumno debe aprender poco a poco los nuevos conceptos, aunque de nuevo es una estrategia que compete al módulo pedagógico decidir, y se escogerá el que menor número de instancias tenga. En caso de que los primeros ejercicios, aquellos con menor número de instancias de conceptos a aprender, tengan el mismo número de instancias, se escogerá uno al azar, pues todos son igualmente recomendables.

2. La resta entre los conjuntos de ejercicios con instancias de conceptos a aprender menos el conjunto de ejercicios con instancias de conceptos prohibidos ha resultado vacía: En este caso los ejercicios se ordenan de menor a mayor según el número de instancias de conceptos prohibidos que posean. Esto es así pues se ha considerado que el ejercicio menos malo será aquel que menos instancias de conceptos prohibidos tenga, ya que no podemos elegir uno sin conceptos prohibidos, y se escogerá el que menor número de instancias tenga. En caso de que los primeros ejercicios, aquellos con menor número de instancias de conceptos prohibidos, tengan el mismo número de instancias, se escogerá uno al azar, pues todos son igualmente recomendables.

Borrado del perfil de la ontología

Tras elegir el ejercicio que se va a recomendar al alumno se procede a borrar el perfil de la ontología. Se borran tanto las instancias genéricas de conceptos,

como la instancia del perfil, como las propiedades que los relacionan. Se borran porque, además de que siempre es bueno borrar lo que no vamos a utilizar más, porque las instancias de conceptos genéricas dependen del conocimiento de cada alumno, y podrían interferir en la recomendación de un nuevo ejercicio a otro alumno.

Devolución del resultado

Este sistema debería devolver el ejercicio a Javy2, para que éste se lo propusiese al jugador. El cómo realizar esta comunicación es algo que no se ha planteado dados los pocos beneficios que aportaba realizar esta comunicación, pues Javy2 todavía no es capaz de coger un ejercicio y proponérselo al usuario. Este módulo le podría devolver el nombre del ejercicio o escribir el ejercicio en un archivo dado para que Javy2 lo leyese. Ambas opciones son válidas, pero actualmente esta comunicación no está realizada, y el módulo Recomendador de Ejercicios muestra por pantalla el ejercicio propuesto al alumno.

5.8. Conclusiones

En este módulo se ha estudiado e implementado un sistema CBR que recomienda el ejercicio más apropiado para cada alumno o usuario de Javy2, en función de su conocimiento y destreza, descritos éstos por el perfil de usuario.

Para ello hemos diseñado e implementado distintos sub-módulos que hemos visto necesarios para llevar a cabo nuestro recomendador. El sistema se compone de los siguientes:

- La Herramienta Java2OWL, elaborada para automatizar la conversión de los ejercicios, seleccionados para formar parte de la base de ejercicios del Javy2, al formato OWL. Este formato nos permite introducir a los ejercicios como instancias de una ontología sobre conceptos de Java. Este módulo, que ya estaba comenzado, nos ha resultado muy importante para conseguir una buena base de ejercicios Java en formato OWL, para mejorar las futuras selecciones.
- Un Editor de perfiles de usuario, para que un usuario externo pueda acceder fácilmente a los perfiles de usuario para crear o modificarlos, mientras que el módulo general no se encuentre integrado en Javy2. Para ello hemos creado el Perfil de Usuario, que almacena información

relevante sobre los ejercicios resueltos por el usuario, así como sobre su experiencia en el juego.

- Un Manejador de perfiles de usuario, que se encarga del tratamiento de los perfiles de usuario como objetos y en formato XML. Trata las inserciones y recuperaciones de perfiles de usuario en XML, ya que será en XML como se guardarán dichos perfiles. Recibe las órdenes del Editor de Perfiles y del Recomendador CBR, y les proporciona la información necesaria sobre dichos perfiles. Esto ha sido necesario para conseguir una mayor modularidad; ya que si en un futuro, en una posible ampliación se quiere cambiar el formato del almacén de perfiles de usuario, sólo sea necesario modificar este sub-módulo, sin alterar los demás.

- ElRecomendador de ejercicios CBR. Sub-módulo principal. Elige el ejercicio más apropiado de la base de conocimiento para cada perfil. Consta de un sub-módulo que se encarga de la inserción, recuperación y borrado de los perfiles como código OWL. El perfil es insertado como un individuo en la ontología sobre conceptos de compilación. En la realización de este sub-módulo hemos estudiado en profundidad las posibles formas de recuperación, eligiendo aquella que nos ha resultado más apropiada.

Capítulo 6

Generador de Escenarios

6.1. Objetivo

Como vimos en el capítulo 5 el recomendador de ejercicios propone al alumno un ejercicio a resolver. Una vez elegido el ejercicio que debe resolver el alumno hay que presentar ese ejercicio para que se pueda resolver en Javy2, es decir, convertirlo a la metáfora de Javy2 que relaciona escenarios del juego con ejercicios. Esta transformación es la que realiza el generador de escenarios, tras proponer un ejercicio de la base de ejercicios, el generador se encargará de transformar el código fuente del ejercicio en un escenario de Javy2.

El generador de escenarios se ocupa de asignar los recursos necesarios para la resolución de un ejercicio propuesto al usuario de Javy2. El generador debe leer dicho ejercicio y extraer todos los recursos que se necesiten para terminar dicho ejercicio. Por ejemplo, si al usuario se le propone un ejercicio que posee la instrucción "bipush 5" el generador debe proporcionar un escenario que contenga al menos un recurso con el valor 5, de otra forma el ejercicio será irresoluble.

Puesto que los ejercicios propuestos difieren según el nivel del usuario es razonable pensar que los recursos proporcionados deberán adecuarse a dicho nivel, por lo que el generador de escenarios será capaz de proponer diferentes escenarios para un mismo ejercicio propuesto a diferentes usuarios, también es razonable pensar que no se deberían repetir escenarios para aumentar el atractivo que el juego pueda tener, por lo que se ha introducido cierto componente aleatorio en la generación para que de resultados diferentes en cada ejecución.

Como podemos ver en la figura 6.1 el generador se encarga de transformar una representación xml del código compilado del ejercicio propuesto en una representación xml de los recursos que dispondrá el escenario.

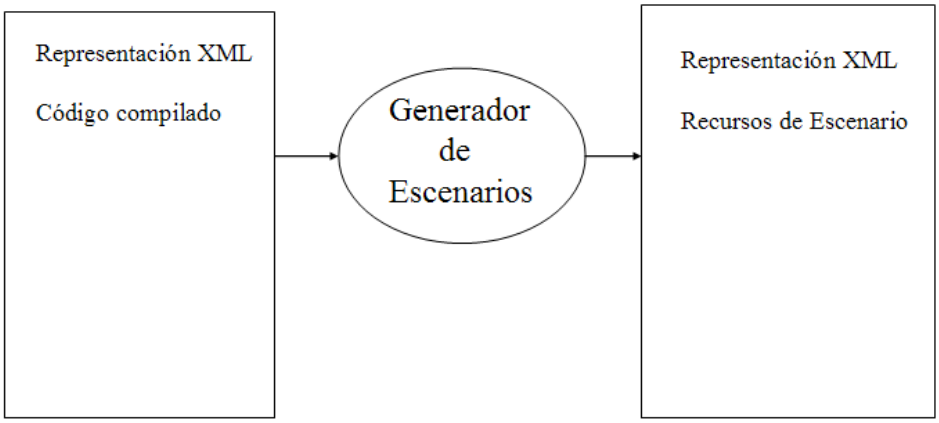

Figura 6.1: Acción del generador de escenarios

Un escenario está formado por varios niveles, cada uno de ellos con unos recursos determinados. Los niveles de los escenarios se corresponden con los métodos de las clases que conforman el ejercicio propuesto. Por lo tanto nuestra representación xml deberá constar de las clases que forman un ejercicio[1], cada una con sus métodos, cada uno de los cuales representa un nivel del escenario, y cada método o nivel estará formado por una lista de recursos propuestos para la resolución de dicho nivel o método. Podemos observar esta estructura en la figura 6.2.

La variación de los escenarios depende exclusivamente del nivel del usuario, en concreto del número total de ejercicios resueltos, puesto que consideramos que todos los recursos, o más concretamente, los bichos que portan estos recursos son igual de difíciles de cazar independientemente del recurso que lleven encima. Hemos tomado el número de ejercicios resueltos como indicador de nivel pues se ha considerado que para medir la capacidad de cazar bichos que porten recursos esta directamente relacionada con el número de

[1]La inclusión de las clases es necesaria para la correcta identificación de los métodos, pues no es lo mismo Square.getArea() que Circle.getArea().

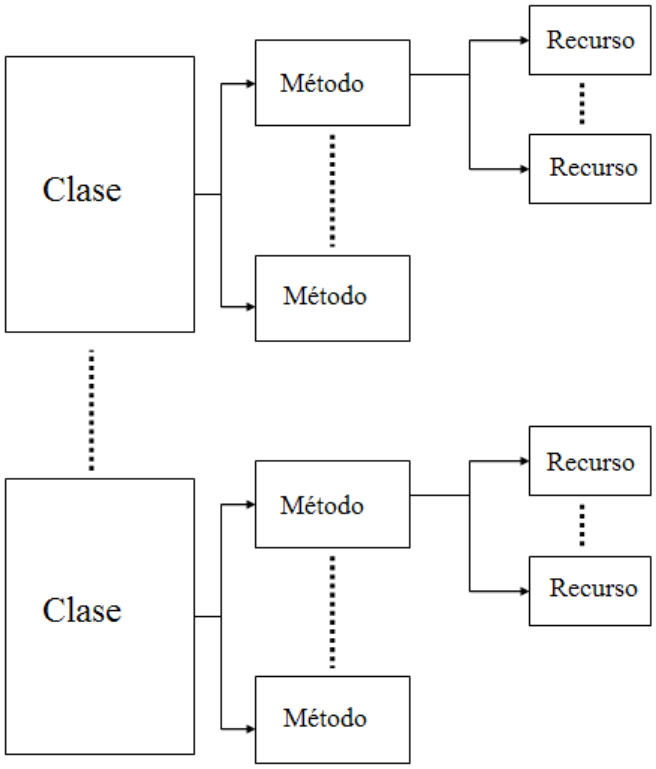

Figura 6.2: Estructura de un escenario

ejercicios que ha resuelto, cuantos más ejercicios haya resuelto, más bichos habrá cazado, y más habilidad tendrá al cazarlos.

6.2. Estructura General

El generador de escenarios está estructurado de la siguiente forma, que podemos ver en la figura 6.3:

1. Obtiene las rutas del archivo xml que define el ejercicio propuesto, del xml donde se generará el escenario, y el nombre del usuario al que se le propone el ejercicio.

2. Analiza el ejercicio extrayendo los recursos mínimos necesarios para resolver dicho ejercicio sin asignar ningún tipo de comportamiento a dichos recursos (todo lo relativo al comportamiento lo veremos más

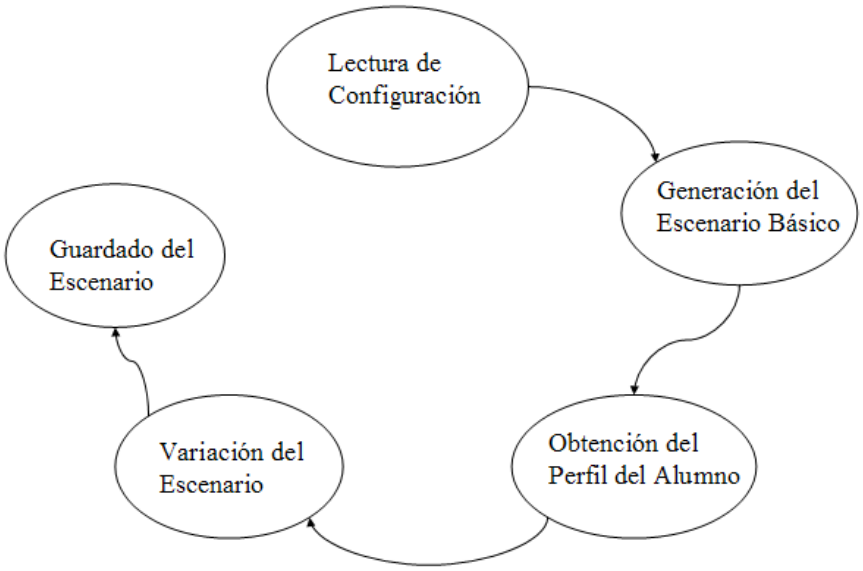

Figura 6.3: Fases del generador de escenarios

adelante). Este primer escenario con los recursos imprescindibles para la resolución del ejercicio es el llamado "escenario básico".

3. Obtiene el perfil del alumno.
4. Varia el escenario básico añadiendo recursos, asignándoles comportamiento, añadiendo recursos sorpresa o introduciendo ruido.
5. Guarda el escenario generado en el xml indicado.

Las siguientes secciones detallan cada uno de estos pasos.

6.2.1. Parámetros de entrada

El generador lee todos sus parámetros de entrada de un archivo de configuración llamado "configGenerador.xml" situado en la ruta "scripts/generador/configGenerador.xml". Dicha ruta no se puede cambiar pues de lo contrario el generador no encontrará los parámetros necesarios para su ejecución y saltará una excepción.

En el archivo configGenerador se encuentran definida la ruta donde se encuentra el ejercicio propuesto, la ruta donde se deberá guardar el escenario generado, y el nombre del usuario al que se le propone el ejercicio.

6.2.2. Análisis del ejercicio. Escenario básico

Una vez obtenida la ruta del archivo donde se encuentra el ejercicio lo abre y lo analiza. Un ejercicio consta de una serie de clases, cada una de ellas con una serie de métodos, y cada método con una serie de instrucciones. El escenario mantiene una estructura similar pues cada nivel del escenario se identifica con un método del ejercicio, pero en vez de instrucciones, cada método o nivel constará de una serie de recursos necesarios para resolver dicho nivel.

Para analizarlo se recorren todas las clases y métodos analizando cada instrucción y, según su código de operación se extraen los parámetros de la instrucción necesarios para generar los recursos requeridos para ejecutar dicha instrucción.

6.2.3. Obtención del perfil del alumno

Una vez obtenido el nombre del usuario se lee el archivo xml de perfiles situado en la ruta obtenida en la lectura de la configuración obteniéndose el perfil del usuario. Dicho perfil consta del número total de ejercicios resueltos, de los conceptos aprendidos, etc.[2] Aunque el dato que nos interesa es el de número de ejercicios resueltos, pues es el que más refleja la habilidad del usuario para cazar recursos.

6.2.4. Reglas de generación. Escenario Variado

Una vez obtenido el perfil y el escenario básico se procede a variar el escenario. Lo primero que se hace es asignar un nivel de habilidad al usuario en función del número de ejercicios resueltos.Dicho nivel se sitúa entre el 0 y 5 y la correspondencia entre el número de ejercicios resueltos y el nivel de habilidad es la siguiente[3]:

Nivel $0 \rightarrow [0, 2)$ ejercicios.

[2]Para más información ver sección sobre perfiles de usuario"

[3]Para cambiar dicha correspondencia se deberían cambiar los límites de los rangos definidos en Escenario.java

Nivel 1 $\rightarrow [2, 6)$ ejercicios.
Nivel 2 $\rightarrow [6, 10)$ ejercicios.
Nivel 3 $\rightarrow [10, 20)$ ejercicios.
Nivel 4 $\rightarrow [20, 50)$ ejercicios.
Nivel 5 $\rightarrow [50, \infty)$ ejercicios.

Una vez obtenido el nivel se procede al duplicado de recursos, para ello se obtiene un rango de probabilidad de duplicar un recurso dependiendo del nivel, como es apreciable observando la correspondencia entre nivel y probabilidad, la probabilidad de duplicar un recurso es mayor cuanto menor sea el nivel del usuario, esto es porque se ha decidido a priori que le resultará más fácil al usuario si hay más recursos disponibles. Se recorren todos los recursos de todos los métodos y para cada recurso se obtiene aleatoriamente una probabilidad real dentro del rango definido para ese nivel y aleatoriamente se decide con esa probabilidad si el recurso se duplica o no.

La correspondencia entre nivel y probabilidad es[4]:

Nivel 0 $\rightarrow [70, 80]$ % probabilidad.
Nivel 1 $\rightarrow [55, 65]$ % probabilidad.
Nivel 2 $\rightarrow [40, 50]$ % probabilidad.
Nivel 3 $\rightarrow [25, 35]$ % probabilidad.
Nivel 4 $\rightarrow [10, 20]$ % probabilidad.
Nivel 5 $\rightarrow [0, 5]$ % probabilidad.

Después de duplicar recursos se procede a añadir recursos sorpresas, estos son recursos que proporcionan una habilidad especial al usuario, como por ejemplo mayor velocidad o mayor eficacia del arma, todos los recursos sorpresas son beneficiosos, por lo que se ha considerado que cuanto menor sea el nivel del usuario más recursos sorpresas debería haber. El número de recursos sorpresas viene dado por un porcentaje sobre el número de recursos totales, y dicho porcentaje viene definido por el nivel del usuario de la siguiente manera[5]:

Nivel 0 $\rightarrow [30, 40]$ % recursos sorpresa.
Nivel 1 $\rightarrow [20, 30]$ % recursos sorpresa.
Nivel 2 $\rightarrow [15, 20]$ % recursos sorpresa.
Nivel 3 $\rightarrow [10, 15]$ % recursos sorpresa.

[4]Para cambiar las probabilidades se debe modificar los métodos probabilidad_minima y probabilidad_maxima de Escenario.java

[5]Si se quiere cambiar dicho porcentaje hay que cambiar el método porcentaje_sorpresas de Escenario.java

Nivel 4 → $[5, 10]$ % recursos sorpresa.
Nivel 5 → $[0, 5]$ % recursos sorpresa.

Una vez obtenido aleatoriamente el porcentaje real de recursos sorpresa (en el rango correspondiente al nivel) se recorren todas las clases y para cada método se obtiene el número de recursos sorpresa (porcentaje de recursos de dicho método). Para cada recurso sorpresa correspondiente al método se decide si finalmente se añade o no según una probabilidad dependiente del nivel (como se ha explicado anteriormente para duplicar los recursos). Una vez decidido que se añade un recurso sorpresa se decide aleatoriamente cual de los 3 tipos de recursos sorpresa se añade[6].

La introducción de ruido es idéntica a la introducción de sorpresas, se obtiene un porcentaje de "recursos ruido" o recursos inútiles dentro de los rangos definidos a continuación[7]:

Nivel 0 → $[0, 10]$ % recursos ruido.
Nivel 1 → $[15, 25]$ % recursos ruido.
Nivel 2 → $[30, 40]$ % recursos ruido.
Nivel 3 → $[45, 55]$ % recursos ruido.
Nivel 4 → $[60, 70]$ % recursos ruido.
Nivel 5 → $[75, 85]$ % recursos ruido.

Para cada método se obtiene el número de recursos ruido, y para cada recurso ruido se decide si finalmente se introduce o no con una probabilidad de *100-Probabilidad de duplicar* puesto que el nivel de dificultad aumentará con el número de recursos ruido.

Para decidir que recurso ruido se añade se elige aleatoriamente el tipo de recurso que se va a introducir (Int,Float,Double,Clase,Método...).Si es un número se genera aleatoriamente mientras que si es una clase,un método o un campo se escoge aleatoriamente de una lista y se le asigna un método aleatoriamente de otra lista[8].

Finalmente se procede a asignar un comportamiento a cada recurso del escenario generado. El comportamiento de los recursos viene definido por 6

[6]Para cambiar los tipos de recursos sorpresa hay que modificar el método genera_sorpresa de Recurso.java

[7]Si se quiere cambiar dicho porcentaje hay que cambiar el método porcentaje_ruido de Escenario.java

[8]Los métodos que generan el ruido son los métodos genera_ruido (el que selecciona el tipo de recurso) y los métodos genera_ruido_XXX. Estos métodos y las listas de las que se escoge están definidas en Recurso.java

atributos. A cada atributo del comportamiento de cada recurso se le asigna un número dependiendo del nivel, los valores que se pueden asignar según el nivel son[9]:

Nivel 0 → [0, 1]
Nivel 1 → [1, 2]
Nivel 2 → [2, 4]
Nivel 3 → [4, 6]
Nivel 4 → [6, 8]
Nivel 5 → [8, 10]

6.2.5. Guardado del escenario generado

Una vez generado el escenario básico y convenientemente variado se procede a guardarse en el xml pasado por parámetro. El formato del xml es semejante al del ejercicio. Consta de una serie de clases con su nombre y una indicación si contienen el método Main, una lista de métodos de cada clase, cada método con su nombre y su tipo, y para cada método una serie de recursos con su valor,su comportamiento definido por las 6 características y su tipo (el tipo está formado por 3 atributos, el primero indica el tipo de dato (Int,Float,Long,Double,String,Surprise), el segundo el tipo de String del que se trata (generic,className,MethodName,field) y el tercero el tipo del método o del campo.

Aquí hay una serie de ejemplos:

```
<recurso valor="Circle" tipo="String" tipoString="className" >
<comportamiento ataque="2" defensa="1" esconderse="1"
                esquivar="1" precavido="1" velocidad="2" />
</recurso>
```

```
<recurso valor="10.0" tipo="Double" >
<comportamiento ataque="1" defensa="2" esconderse="2"
                esquivar="2" precavido="1" velocidad="2" />
</recurso>
```

```
<recurso valor="getArea" tipo="String" tipoString="MethodName"
tipoMetodo="()D" >
<comportamiento ataque="1" defensa="1" esconderse="2"
                esquivar="1" precavido="2" velocidad="1" />
```

[9]Para cambiar estos valores hay que modificar el método get_comportamiento_nivel de Escenario.java

```
</recurso>
```

```
<recurso valor="_radius" tipo="String" tipoString="field"
tipoMetodo="D" >
<comportamiento ataque="1" defensa="2" esconderse="2"
                esquivar="1" precavido="2" velocidad="2" />
</recurso>
```

6.3. Validación

Todas las reglas explicadas anteriormente han sido establecidas a priori, por la experiencia o sentido común de los desarrolladores. Esto no es deseable, pues la concepción que se tenía a priori puede ser muy equivocada, por esto mismo normalmente se realizan diversas pruebas que permiten afinar el sistema a lo que el usuario realmente necesita. En nuestro caso no se han podido realizar pruebas para validar el sistema ya que el juego Javy2 todavía esta en proceso de desarrollo. Cuando se finalice el juego, se podrá proponer diversos escenarios a usuarios de determinado nivel, y comprobar si el escenario propuesto ha sido adecuado al nivel del usuario, y en caso negativo se podrán ajustar los parámetros para ajustar correctamente las reglas.

6.4. Mantenimiento

En caso de necesitar modificar la estructura de directorios se debe intentar conservar la ruta del archivo de configuración, y modificar su contenido. Si es estrictamente necesario se puede definir la nueva ruta del archivo de configuración en la clase GeneradorEscenarios.java.

6.5. Ampliaciones

Puesto que no se ha podido evaluar el sistema no podemos decir si este es bueno o malo, o si necesitará muchas ampliaciones.

Como posible ampliación está la inclusión de nuevas reglas para variar el escenario, así como la modificación de las reglas actuales, parámetros, número de niveles, etc.

También podría ser deseable que el nivel del usuario no se estableciese únicamente mediante el número de ejercicios resueltos, si no por ejemplo por un indicador que llevase el número de ejercicios resueltos ponderados por su dificultad. Esto conllevaría una mejor clasificación del usuario en un nivel determinado.

En caso de asignar bichos con diferentes características, por ejemplo de velocidad, independientes del comportamiento a los diferentes tipos de recursos, por ejemplo asignar un bicho de velocidad 1 a los valores enteros, mientras que asignamos un bicho de velocidad 7 a las clases, entonces las reglas deberían tener en cuenta el tipo de recurso. Además habría que establecer alguna forma de medir la habilidad del usuario al cazar bichos de uno u otro tipo.

6.6. Conclusiones

Dentro de un sistema de enseñanza interactivo hemos separado dos importantes tareas, la proposición de un ejercicio adecuado al alumno, y la presentación de dicho ejercicio de una forma entretenida al alumno. Ambas partes son muy importantes, pues si no se proporciona un ejercicio adecuado el método de aprendizaje estará fallando perjudicando la capacidad de aprender del alumno, y si falla la presentación del ejercicio el alumno se aburrirá antes de completarlo quedando su aprendizaje incompleto.

La proposición de un ejercicio adecuado corre a cargo del módulo pedagógico de Javy2, en el que estaría integrado el recomendador de ejercicios, mientras que la presentación del ejercicio estaría a cargo del propio juego de Javy2, en concreto de su metáfora.

El generador de escenarios se encarga de unir ambas tareas, proporcionando un método para transformar el ejercicio propuesto por el recomendador de ejercicios en un escenario de la metáfora de Javy2, reforzando además la capacidad de entretenimiento de la metáfora ajustando el nivel de dificultad del escenario al nivel de habilidad del usuario.

Capítulo 7

Conclusiones

Este proyecto forma parte de un sistema de enseñanza interactivo. Estos sistemas siguen la metodología de aprendizaje de "learning by doing", que está llamada a sustituir a la metodología tradicional de un profesor explicando sus conocimientos a los alumnos. Los sistemas de enseñanza interactivos facilitan al alumno la tarea de adquisición de conocimiento, redundando además en un conocimiento mejor y más duradero. Estos sistemas, para que sean prácticos deben estar acompañados de explicaciones sobre lo que el alumno está realizando, de forma que se puedan resolver las dudas que le surjan al alumno, es aquí donde nuestro sistema se sitúa, proporcionando al usuario un sistema de ayuda fácil e intuitivo capaz de solucionar la mayoría de las dudas que le puedan surgir al aprender la compilación de Java.

Los módulos desarrollados son una parte crucial del sistema de enseñanza interactivo, no sólo por la capacidad de resolver dudas al alumno, también por la necesidad de proponer ejercicios adecuados al conocimiento del alumno. La necesidad de que el alumno aprenda ciertas materias antes que otras, en un aprendizaje secuencial, ha sido plenamente aceptada en el estudio de las metodologías de aprendizaje, y ya nadie enseña a multiplicar antes que a sumar. Debido a esta necesidad y a la exigencia de que el alumno, como jugador, mantenga su interés, sin perderlo porque le resulta demasiado fácil o difícil un ejercicio, la proposición de un ejercicio adecuado al conocimiento del alumno es una parte fundamental de cualquier sistema de enseñanza.

La conversión de los problemas presentados al alumno en un escenario de Javy2 es evidentemente imprescindible, puesto que sin esta conexión no tendríamos un juego educativo, y no podríamos aplicar ninguna metodología de aprendizaje al juego, quedando este incapacitado para la enseñanza.

En este proyecto se ha tratado la información mediante dos diferentes aproximaciones, la sintáctica y la semántica. En la aproximación sintáctica se han tratado los textos por su sintaxis y su estructura, mientras que en la aproximación semántica se han tratado los textos por los significados de sus palabras. Ambas aproximaciones tienen sus pros y sus contras, y en ocasiones es mejor aplicar una u otra, aunque debido a la inmensa complejidad del lenguaje ninguna de las dos obtendrá un resultado perfecto. Probablemente la mejor forma de tratar los textos sea una aproximación combinada que tenga en cuenta tanto la sintaxis de los textos como su significado.

Hay que mencionar que en este proyecto se han cumplido gran parte de lo inicialmente propuesto. Pese a que en un principio nos propusimos implementar el comportamiento de los personajes no jugadores de Javy2, al no ser esto posible por causas ajenas a nosotros, nos centramos en el aspecto más pedagógico del juego, que ha terminado por ser una parte muy importante del mismo. Además al centrarnos en el aspecto pedagógico descubrimos la necesidad de convertir los ejercicios Java en una representación semántica, realizando una herramienta de traducción a esta representación, cuya importancia trasciende los límites del juego Javy2.

La experiencia de trabajar con un grupo de investigación como GAIA ha sido muy satisfactoria, no sólo por todos los conocimientos adquiridos, también porque se ha aprendido a trabajar dentro de un proyecto grande y con un gran grupo, lo cual resultará tremendamente útil en el futuro.

Apéndice A

Apéndice de la Ayuda Sintáctica

A.1. Signatura de los métodos Java invocados desde C++

La codificación de los tipos de Java en forma de cadena se muestran en la tabla A.1.

Signatura	Tipo de Java
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
Lpaquete/paquete/nombre_clase;	clase
[signatura	tipo[]
(signaturas)tipo_devuelto	tipo_ devuelto (signaturas) (función)

Cuadro A.1: Codificación de tipos de Java en cadenas.

A.2. Sistema de archivos de la ayuda sintáctica

Configuración de los paquetes de jCOLIBRI

Es necesario proporcionar la ruta del directorio donde están definidos los diferentes paquetes que forman jCOLIBRI.
Clase: *jcolibri.cbrcore.packageManager.PackageFinder*
Método: *findPackages()*
Ruta Actual: *"scripts/config"*

Configuración del conector y la estructura de los casos

Es necesario proporcionar la ruta de los archivos donde están definidos los conectores y la estructura de los casos. En nuestro caso se usan dos módulos independientes, uno para la ayuda sobre la jvm y otro sobre la ayuda del juego, por lo que tenemos dos conectores y dos estructuras de casos diferentes.

- Conectores:

Clase: *help.jvm.english.textual.TextualCBR*
Método: *Constante de clase*
Ruta Actual: *".\scripts\configCBRTextual\connector.xml"*
Clase: *help.game.english.textual.TextualGame*
Método: *Constante de clase*
Ruta Actual: *".\scripts\configGameTextual\connector.xml"*

- Estructura de casos:

Clase: *help.jvm.english.textual.TextualCBR*
Método: *Constante de clase*
Ruta Actual: *".\scripts\configCBRTextual\caseStructure.xml"*
Clase: *help.game.english.textual.TextualGame*
Método: *Constante de clase*
Ruta Actual: *".\scripts\configGameTextual\caseStructure.xml"*

Modificación de los archivos connector.xml

Es necesario modificar los archivos connector.xml puesto que estos incluyen las rutas donde se encuentran los archivos de estructura de casos y contenido de los mismos(los textos).

• Conector JVM:
Elemento xml: */Connector/CaseStructure*
Ruta Actual: *".\scripts\configCBRTextual\caseStructure.xml"*
Elemento xml: */Connector/OTHER/Parameter/ParameterValue*
Ruta Actual: *".\scripts\configCBRTextual\JVM.txt"*

• Conector Game:
Elemento xml: */Connector/CaseStructure*
Ruta Actual: *".\scripts\configGameTextual\caseStructure.xml"*
Elemento xml: */Connector/OTHER/Parameter/ParameterValue*
Ruta Actual: *".\scripts\configGameTextual\game.txt"*

Archivos de reglas

Es necesario que el sistema pueda encontrar los archivos donde están definidas las reglas para el procesamiento del texto. Cada módulo necesita conocer donde están estos archivos.

• Módulo de ayuda de la JVM:
Clase:*help.jvm.english.textual.TextualCBR*
Método:*Constante de clase FEATURERULES_FILE*
Ruta actual:*scripts\configCBRTextual\featuresRules.txt*
Clase:*help.jvm.english.textual.TextualCBR*
Método:*Constante de clase DOMAINRULES_FILE*
Ruta actual:*scripts\configCBRTextual\DomainRules.txt*
Clase:*help.jvm.english.textual.TextualCBR*
Método:*Constante de clase PHRASESRULES_FILE*
Ruta actual:*scripts\configCBRTextual\phrasesRules.txt*
Clase:*help.jvm.english.textual.TextualCBR*
Método:*Constante de clase GLOSSARY_FILE*
Ruta actual:*scripts\configCBRTextual\glossary.txt*

• Módulo de ayuda del juego javy2:
Clase:*help.game.english.textual.TextualGame*
Método:*Constante de clase FEATURERULES_FILE*
Ruta actual:*scripts\configCBRTextual\featuresRules.txt*
Clase:*help.game.english.textual.TextualGame*
Método:*Constante de clase DOMAINRULES_FILE*
Ruta actual:*scripts\configCBRTextual\DomainRules.txt*
Clase:*help.game.english.textual.TextualGame*
Método:*Constante de clase PHRASESRULES_FILE*

Ruta actual:*scripts\configCBRTextual\phrasesRules.txt*
Clase:*help.game.english.textual.TextualGame*
Método:*Constante de clase GLOSSARY_FILE*
Ruta actual:*scripts\configCBRTextual\glossary.txt*

Archivo de configuración de JWNL

Para hacer uso del diccionario incorporado JWNL es necesario establecer la ruta para el archivo en el que está su configuración.Esto se hace en:
Clase:*jcolibri.extensions.textual.method.WordNetRelationsMethod*
Método:*Constante de clase config_path*
Ruta actual: *"scripts\configCBRTextual\jwnl\file_properties.xml"*

Contenido del archivo de configuración de JWNL.

En el archivo de configuración se ha de poner la ruta donde está el diccionario JWNL.
Archivo:*file_properties.xml*
Elemento xml:*<param name="dictionary_path" value=.../>atributo value*
Ruta actual: *"scripts\examples\TextualCBR\dict"*

Apéndice B

Apéndice de la Ayuda Semántica

B.1. Estructura general

La ayuda semántica se compone de dos módulos o paquetes: **help.game** y **motor**. **help.game** contiene la interfaz **GameGenericHelp**, que será la que se comunique con Javy2.

B.1.1. El paquete motor

El módulo se controla por una clase principal, llamada **AyudaSemantica**, que implementa el interfaz **GameGenericHelp**. Esta clase es la encargada de la inicialización del sistema, de su funcionamiento e interrupción. Se encarga de la comunicación con el resto de módulos o clases que conforman el sistema. Estas clases son:

- **InteracciónOntobrige**: Comunica la ayuda semántica con Ontobridge, de forma que todo el manejo de ontologías se hace desde aquí. De esta forma conseguimos una mayor modularidad para facilitar posibles cambios futuros.
- **FiltraPreguntas**: Encargado del tratamiento de la pregunta o query en lenguaje natural. Contiene los métodos necesarios para filtrar símbolos y palabras vacías, y para tratar los plurales.
- **Quicksort**: Ordena la lista de respuestas por orden de prioridad.
- **MainPrueba**: Clase con el main para probar el sistema por consola.

La clase AyudaSemantica

En esta sección se explicarán los componentes de la clase, así como el funcionamiento del método de solicitud de ayuda.

La clase AyudaSemantica consta de los siguientes atributos:

- **String pregunta**: Contiene la query o pregunta del usuario. Esta query será tratada para conservar sólo las palabras que puedan ser útiles para buscar las similitudes.
- **ArrayList<String>lista**: Una vez que la pregunta es tratada, se guarda en este array como una lista de palabras, ya útiles.
- **Hashtable<String, ArrayList>conceptos**: Conceptos es una tabla Hash cuyas claves son los nodos de la ontología, y el valor es un ArrayList con los valores de sus etiquetas; es decir, los conceptos relacionados con cada nodo. Esta tabla se rellena en el preciclo, para no necesitar acceder a la ontología para cada palabra de la query.
- **InteraccionOntobridge interacOb**: Objeto que realiza cualquier operación con Ontobridge.
- **ArrayList<String>listaRelacionados**: Lista con los nombres de los nodos de la ontología relacionados con la query.
- **ArrayList<Float>nivelVerosimilitud**: Lista asociada a listaRelacionados. Si en la posición i de listaRelacionados se encuentra el nombre del nodo Nodo-i, en la posición i de nivelVerosimilitud se encontrará el nivel de similitud del concepto de la ontología de nombre Nodo-i con la query.
- **int numSol**: Número de conceptos relacionados encontrados. Su valor inicial será 0.

Ahora se explicarán los métodos más importantes de la clase:

El método public void init()

Inicializa la ayuda semántica. Para ello inicia el objeto interacOb y con él rellena el HashTable conceptos. Inicializa también los ArrayList.

El método public String help(String question)

Si la pregunta introducida es correspondiente a un "more like this"busca el siguiente texto por similitud y lo devuelve. Si no:

1. Filtra la query y elimina las palabras vacías.
2. Obtiene las respuestas con similitud semántica. Si no hay similares, devuelve "I don't understand you...". Si no:
3. Las ordena por nivel de similitud.
4. Devuelve la de mayor prioridad

El método public double getPositionSimilitude(int pos)

Devuelve la similitud del elemento pos del vector de resultados. Si pos está fuera de rango devuelve -1.

public String getPositionHelp(int pos)

Devuelve la ayuda del elemento pos del vector de resultados. Si pos está fuera de rango devuelve null.

public static boolean is_more_like_this(String query)

Devuelve cierto si la query se reconoce como una "more like this"

private void filtraPregunta()

1. Elimina los símbolos que puede haber en la query
2. Elimina las palabras vacías
3. Trata los plurales

public void close() throws Exception

Cierra la aplicación.

public double getBestSimilitude()

Devuelve la mejor similitud obtenida.

La clase FiltraPreguntas

La clase FiltraPreguntas consta de los siguientes atributos estáticos:

- **signs**: Una lista de caracteres con todos los signos que deberán eliminarse de la pregunta del usuario.
- **stopWords**: Lista de palabras vacías. Un ejemplo de palabras vacías que nuestro sistema elimina de la pregunta: “allows”, “almost”, “alone”, “along”, “already”, “also”, “although”, “always”, “am”, “among”, “amongst”, “an”, “and”, “another”, “any”, “anybody”, “anyhow”, “anyone”, “anything”, “anyway”, “anyways”, “anywhere”, “apart”, “appear”, “appreciate”, “appropriate”, “are”, “aren’t”, “as”.

Métodos principales:

public static ArrayList<String>filtraSimbolos(String pregunta)

Recibe como entrada el String con la pregunta del usuario, y la separa por palabras, eliminando los signos que se va encontrando. La salida es una lista de Strings. Cada elemento de la lista es una palabra.

public static ArrayList<String>filtra(ArrayList<String>pregunta)

Recorre elemento a elemento la lista que compone la pregunta, y comprobando las stopWords, elimina las palabras vacías.

public static ArrayList<String>arreglaPlurales(ArrayList<String>pregunta)

Devuelve la lista de palabras de la pregunta junto con los singulares de los posibles plurales.

La clase InteraccionOntobridge

La clase InteraccionOntobridge sólo contiene un atributo:

- **Ontobridge ob**, para conectar el sistema con el Ontobridge.

Métodos principales:

public InteraccionOntobridge()

Constructor. Inicializa ob y carga las ontologías y subontologías.

public Iterator<String>getAllClasses()

Devuelve un iterador con los nombres de todas las clases de la ontología.

public Iterator<String>getParents(String concept)

Devuelve una lista Iterator con los padres del concepto con nombre concept.

public boolean isLeaf(String concept)

Comprueba si el concepto de nombre concept es hoja en la ontología.

public ArrayList<String>getLabelValue(String nombreConcepto)

Devuelve los valores del atributo label de la clase cuyo nombre pasa como parámetro. Si la clase no tiene atributos devuelve un iterador vacío.

public Hashtable<String,ArrayList>getHashClassLabel()

Devuelve una tabla Hash cuyos elementos son:

- Clave: nombre de la clase (el concepto). (String)
- Valor: lista de sinónimos. (Iterator <String>)

public String getText(String nombreConcepto)

Devuelve el texto del fichero asociado a la clase de la ontología de nombre nombreConcepto. Este texto se convierte a String.

public String getShortName(String s)

Devuelve el "nombre corto"de la clase de la ontología de nombre "largo" s.

Apéndice C

Apéndice de la Evaluación de las Ayudas

Evaluación realizada por el Evaluador número seis

Realizó 20 preguntas a cada una de las ayudas puntuando cada respuesta en una escala del 0 al 10, en nuestra evaluación consideramos una respuesta valorada a partir de 6 como respuesta correcta. También nos proporcionó su respuesta esperada:

Pregunta 1: what is JVM?

Respuesta ayuda semántica: 0

No hay conceptos relacionados Tiempo tardado por ayuda semántica: 250 ms

Respuesta ayuda sintáctica: 4

How to get to the instruction selection Help: When you want to execute a JVM instruction, you only have to write it down in the JVM console. Reach the console moving your avatar towards it in the virtual environment, and press the 'use' key. Tiempo de respuesta: 2594 ms

Respuesta esperada: la respuesta de la ayuda sintactica a la pregunta every person in the game is good or bad? or can be neutral people ? (ver más abajo) Sería adecuada como respuesta a esta pregunta.

Pregunta 2: what are boxes for?

Respuesta ayuda semántica: 10

*CONCEPTO: Box

*DESCRIPCION: The Boxes represent storage.We have one type of box for every datatype of theJVM. They are distinguished through a letter (f, i, . . .) on every of its sides.To know

the contents of a box the player must click on it with the mouse,while holding the button a semi-transparent layer appears in the HUD withthe value of the resource inside of the box.

Tiempo tardado por ayuda semántica: 141 ms

Respuesta ayuda sintáctica: 0

I don´t understand you... Tiempo de respuesta: 281 ms

Pregunta 3: what is the meaning of the f that appears in this box

Respuesta ayuda semántica: 10

*CONCEPTO: Box

*DESCRIPCION: The Boxes represent storage.We have one type of box for every datatype of theJVM. They are distinguished through a letter (f, i, . . .) on every of its sides.To know the contents of a box the player must click on it with the mouse,while holding the button a semi-transparent layer appears in the HUD withthe value of the resource inside of the box.

Tiempo tardado por ayuda semántica: 172 ms

Respuesta ayuda sintáctica: 0

The concept you are looking for is: Enemies Description Help: The avatars that populate the environment hide resources that you may need in order to execute the Java code. Initially, you do not know which resource hide every avatar. Therefore, if you are looking for an specific resource, you will have to use trial and error, stealing the resources of the avatars, and checking whether they are what you were looking for or not. Eventually, you will see some patterns between the appearance of the avatars and the resources they hide.

Tiempo de respuesta: 500 ms

Pregunta 4: how can I see what is inside of a box?

Respuesta ayuda semántica: 10

*CONCEPTO: Box

*DESCRIPCION: The Boxes represent storage.We have one type of box for every datatype of theJVM. They are distinguished through a letter (f, i, . . .) on every of its sides.To know the contents of a box the player must click on it with the mouse,while holding the button a semi-transparent layer appears in the HUD withthe value of the resource inside of the box.

Tiempo tardado por ayuda semántica: 140 ms

Respuesta ayuda sintáctica: 0

I don´t understand you... Tiempo de respuesta: 359 ms

Pregunta 5: do i have x ray?

Respuesta ayuda semántica: 0

No hay conceptos relacionados

Tiempo tardado por ayuda semántica: 31 ms

Respuesta ayuda sintáctica: 0 I don´t understand you... Tiempo de respuesta: 265 ms

---

Pregunta: who are the enemies ?

Respuesta ayuda semántica: 9 (la respuesta es la esperada pero el texto está redactado un poco raro.. por eso en vez de 10 he puesto 9)

*CONCEPTO: Enemies_localization

*DESCRIPCION: Characters with resources populate the environment. Therefore, you may find them all over the game. Though you sometimes can see them on their own roaming, usually they like to flock together, to make themselves stronger. Try to isolate one before attacking it to steal the resource it hides.

Tiempo tardado por ayuda semántica: 63 ms

Respuesta ayuda sintáctica: 9

Enemies localization Help: Characters with resources populate the environment, those are your enemies. Therefore, you may find them all over the game. Though you sometimes can see them on their own roaming, usually they like to flock together, to make themselves stronger. Try to isolate one before attacking it to steal the resource it hides.

Tiempo de respuesta: 781 ms

Pregunta: how can I recognize my enemies?

Respuesta ayuda semántica: 7 La respuesta esperada que querría saber si hay algún rasgo físico externo que me ayude a distinguir a los enemigos de otros posibles compañeros de equipo, o del otro equipo porque esto es un entorno multijugador.

*CONCEPTO: Enemies_localization

*DESCRIPCION: Characters with resources populate the environment. Therefore, you may find them all over the game. Though you sometimes can see them on their own roaming, usually they like to flock together, to make themselves stronger. Try to isolate one before attacking it to steal the resource it hides.

Tiempo tardado por ayuda semántica: 62 ms

Respuesta ayuda sintáctica: 7

The concept you are looking for is: Enemies localization Help: Characters with resources populate the environment, those are your enemies. Therefore, you may find them all over the game. Though you sometimes can see them on their own roaming, usually they like to flock together, to make themselves stronger. Try to isolate one before attacking it to steal the resource it hides.

Pregunta: every person in the game is good or bad? or can be neutral people ?

Respuesta ayuda semántica: 5

*CONCEPTO: Enemies_Graphics

*DESCRIPCION: The avatars that populate the environment hide resources that you may need in order to execute the Java code. Initially, you do not know which resource hide every avatar. Therefore, if you are looking for an specific resource, you will have to use trial and error, stealing the resources of the avatars, and checking whether they are what you were looking for or not. Eventually, you will see some patterns between the appearance of the avatars and the resources they hide.

Tiempo tardado por ayuda semántica: 78 ms

Respuesta ayuda sintáctica: 0

The concept you are looking for is: Final target of the game Help: The goal or target of the game is to learn how the java virtual machine compiles the different java programs. In order to reach this goal you will be proposed different java exercises that you should compile. To compile an exercise you will need to capture criatures that carry some resources and know how and when to use those resources in order to complete the exercise, that is to translate the java source code into the JVM opcode (compile it).Finishing exercises you will be gaining knowledge in different compilation areas. Once you master all the categories of knowledge you will become the MASTER OF JVM, the game will have nothing to teach tou and you will win the game and it will finish.

Tiempo de respuesta: 375 ms

Pregunta: more

Respuesta ayuda semántica: 2

*CONCEPTO: User_movement

*DESCRIPCION: The user controls the movement of his player or avatar with the keys shown below:W : Moves forward.S : Moves backward.A : Left strafe. Turn to the left.D : Right strafe. Turn to the right.Remember movement is understood from the point of view of the avatar, it does notmatter where the camera is looking. This way if the camera is looking to the face ofthe avatar and you move forward the avatar will move to the camera.You can move the camera with the keyboard arrows.To interact with another NPC's or things you should press the U key.

Tiempo tardado por ayuda semántica: 0 ms

Respuesta ayuda sintáctica: 0

The concept you are looking for is: Use of Help Help: Hello!, I am Help, and my task is to solve any doubt you can have about JVM or the game.

Tiempo de respuesta: 0 ms

Pregunta: more

Respuesta ayuda semántica: 2

*CONCEPTO: Enemies_localization

*DESCRIPCION: Characters with resources populate the environment. Therefore, you may find them all over the game. Though you sometimes can see them on their own roaming, usually they like to flock together, to make themselves stronger. Try to isolate one before attacking it to steal the resource it hides.

Tiempo tardado por ayuda semántica: 15 ms

Respuesta ayuda sintáctica: Esta no la he hecho.

Pregunta: more Respuesta ayuda semántica

*CONCEPTO: Final_target_of_the_game

*DESCRIPCION: The goal or target of the game is to learn how the java virtual machine compiles the different java programs. In order to reach this goal you will be proposed different java exercises that you should compile. To compile an exercise you will need to capture criatures that carry someresources and know how and when to use those resources in order to complete the exercise, that is to translate the java source code into the JVM opcode (compile it).Finishing exercises you will be gaining knowledge in different compilation areas. Once you master all the

categories of knowledge you will become the MASTER OF JVM, the game will have nothing to teach tou and you will win the game and it will finish.

Tiempo tardado por ayuda semántica: 31 ms

Respuesta ayuda sintáctica: Esta no la he hecho.

Pregunta: when I win?

Respuesta ayuda semántica: 0

No hay conceptos relacionados

Tiempo tardado por ayuda semántica: 15 ms

Respuesta ayuda sintáctica: 6

The concept you are looking for is: Final target of the game Help: The goal or target of the game is to learn how the java virtual machine compiles the different java programs. In order to reach this goal you will be proposed different java exercises that you should compile. To compile an exercise you will need to capture criatures that carry some resources and know how and when to use those resources in order to complete the exercise, that is to translate the java source code into the JVM opcode (compile it).Finishing exercises you will be gaining knowledge in different compilation areas. Once you master all the categories of knowledge you will become the MASTER OF JVM, the game will have nothing to teach tou and you will win the game and it will finish.

Tiempo de respuesta: 359 ms

Pregunta: how do I know I've finished the game?

Respuesta ayuda semántica: 1

*CONCEPTO: User_movement

*DESCRIPCION: The user controls the movement of his player or avatar with the keys shown below:W : Moves forward.S : Moves backward.A : Left strafe. Turn to the left.D : Right strafe. Turn to the right.Remember movement is understood from the point of view of the avatar, it does notmatter where the camera is looking. This way if the camera is looking to the face ofthe avatar and you move forward the avatar will move to the camera.You can move the camera with the keyboard arrows.To interact with another NPC's or things you should press the U key.

Tiempo tardado por ayuda semántica: 46 ms

Respuesta ayuda sintáctica: 10

The concept you are looking for is: Final target of the game Help: The goal or target of the game is to learn how the java virtual machine compiles the different java programs. In order to reach this goal you will be proposed different java exercises that you should compile. To compile an exercise you will need to capture criatures that carry some resources and know how and when to use those resources in order to complete the exercise, that is to translate the java source code into the JVM opcode (compile it).Finishing exercises you will be gaining knowledge in different compilation areas. Once you master all the categories of knowledge you will become the MASTER OF JVM, the game will have nothing to teach tou and you will win the game and it will finish.

Tiempo de respuesta: 484 ms

Pregunta: what weapons can I use?

Respuesta ayuda semántica: 10

*CONCEPTO: Weapon

*DESCRIPCION: The weapon in the game is called resource hunter.Each opponent has only one weapon, it doesn't use any type of ammunition neither consumes energy. The resource hunter allows the player to hunt resource carriers. The Resource hunter may only be used in the outsides of each level.

Tiempo tardado por ayuda semántica: 47 ms

Respuesta ayuda sintáctica: 10

The concept you are looking for is: Weapon Help: The weapon in the game is called resource hunter. Each opponent has only one weapon, it doesn't use any type of ammunition neither consumes energy. The resource hunter allows the player to hunt resource carriers. The Resource hunter may only be used in the outsides of each level.

Tiempo de respuesta: 250 ms

Pregunta: how can I get more weapons?

Respuesta ayuda semántica: 7

*CONCEPTO: Weapon

*DESCRIPCION: The weapon in the game is called resource hunter.Each opponent has only one weapon, it doesn't use any type of ammunition neither consumes energy. The resource hunter allows the player to hunt resource carriers. The Resource hunter may only be used in the outsides of each level.

Tiempo tardado por ayuda semántica: 46 ms

Respuesta ayuda sintáctica: 7

The concept you are looking for is: Weapon Help: The weapon in the game is called resource hunter. Each opponent has only one weapon, it doesn't use any type of ammunition neither consumes energy. The resource hunter allows the player to hunt resource carriers. The Resource hunter may only be used in the outsides of each level.

Tiempo de respuesta: 281 ms

Pregunta: How can I use the elevator ?

La pregunta está relacionada con el uso y sin embargo la ayuda semántica no lo detecta, pero la sintáctica sí.

Respuesta ayuda semántica: 6

*CONCEPTO: Meaning_of_the_elevator

*DESCRIPCION: Elevator: when a player executes an instruction of bytecode that supposes the creation of frame, its elevator of ascentwill be activated and will be able to cross it to arrive to a new level.When an instruction is executed that supposes the conclusion of frame present, its elevator of slope will be activated,that will take to the level from which it came. An activated elevator will have sobreimposed a label indicating frame to which it directs (name of the invoked method). Any player will be able to see this label.

Tiempo tardado por ayuda semántica: 47 ms

Respuesta ayuda sintáctica: 10

The concept you are looking for is: Use of the elevator Help: When a user executes an instruction that supposes the creation of a new frame (for example invokevirtual) his elevator will be activated and he will be able to go up to reach a new level. When an instruction that supposes the end of the actual frame is executed (for example return) his elevator will be activated and the user will be able to go down to the level he came from. An activated lift has text showing the frame it goes to (name of the method).

Tiempo de respuesta: 359 ms

Pregunta: what are rooms?

Querría saber qué son y para qué se usan más o menos se contesta la pregunta. Quiero saber la correspondencia de las rooms con la JVM.

Respuesta ayuda semántica: 7

*CONCEPTO: Stealing_from_the_security_room

*DESCRIPCION: In every control room there is a security room or vault to protect the resources of the player when he changes his level. The security room stays closed until the user comes again or a certain time has passed.When it opens because of the timeout a message is send to all the players of the game, and the first one who arrives to the security room takes control of all the resources that where in it.

Tiempo tardado por ayuda semántica: 15 ms

Respuesta ayuda sintáctica: 7

The concept you are looking for is: Stealing from the security room Help: In every control room there is a security room or vault to protect the resources of the player when he changes his level. The security room stays closed until the user comes again or a certain time has passed. When it opens because of the timeout a message is send to all the players of the game, and the first one who arrives to the security room takes control of all the resources that where in it.

Tiempo de respuesta: 500 ms

Pregunta: where are rooms? Realmente no contesta a la pregunta de dónde están pero sí habla de las rooms.

Respuesta ayuda semántica: 6

*CONCEPTO: Stealing_from_the_security_room

*DESCRIPCION: In every control room there is a security room or vault to protect the resources of the player when he changes his level. The security room stays closed until the user comes again or a certain time has passed.When it opens because of the timeout a message is send to all the players of the game, and the first one who arrives to the security room takes control of all the resources that where in it.

Tiempo tardado por ayuda semántica: 47 ms

Respuesta ayuda sintáctica: 6

The concept you are looking for is: Stealing from the security room Help: In every control room there is a security room or vault to protect the resources of the player when he changes his level. The security room stays closed until the user comes again or a certain time has passed. When it opens because of the timeout a message is send to all the players of the game, and the first one who arrives to the security room takes control of all the resources that where in it.

Tiempo de respuesta: 297 ms

Pregunta: more

No hay mas conceptos relacionados

Tiempo tardado por ayuda semántica: 0 ms

Respuesta ayuda sintáctica: No la he hecho.

Pregunta: Is there any place where I can consult which rooms do I have in the ship?

Aquí la ayuda sintáctica va igual que antes reconoce una pregunta sobre rooms.. pero la semántica no sabe identificar el tema.

Respuesta ayuda semántica: 0

*CONCEPTO: Mentors_Localization

*DESCRIPCION: The localization of the mentors changes from one exercise to another, but thy always remain near the areas they act. For example a mentor that is able to execute the invokevirtual instruction will be near the elevators, because that instruction activates them.

Tiempo tardado por ayuda semántica: 109 ms

Respuesta ayuda sintáctica: 6

The concept you are looking for is: Stealing from the security room Help: In every control room there is a security room or vault to protect the resources of the player when he changes his level. The security room stays closed until the user comes again or a certain time has passed. When it opens because of the timeout a message is send to all the players of the game, and the first one who arrives to the security room takes control of all the resources that where in it.

Tiempo de respuesta: 328 ms

Pregunta: more

Respuesta ayuda semántica: 0

No hay mas conceptos relacionados

Tiempo tardado por ayuda semántica: 0 ms

Respuesta ayuda sintáctica: 6

The concept you are looking for is: Meaning of the security room Help: In each control room a security room for each opponent will exist. Its purpose will be to protect the resources of a player by a certain time when this one changes of level.

Tiempo de respuesta: 0 ms

Apéndice D

Apéndice de la Herramienta Java2Owl

Elementos implementados de la DTD de conceptos de Java

Los siguientes elementos se corresponden con los que se estimaron necesarios para poder convertir a OWL los ejercicios Java de tipo imperativo.

```
<!ELEMENT literal-null EMPTY>

<!ELEMENT literal-string EMPTY>
<!ATTLIST literal-string
                value CDATA #REQUIRED>


<!ELEMENT literal-char EMPTY>
<!ATTLIST literal-char
        value CDATA #REQUIRED>


<!ELEMENT literal-number EMPTY>
<!ATTLIST literal-number
                value CDATA #REQUIRED
                    %kind-attribute;
                 base CDATA "10">


<!ELEMENT literal-boolean EMPTY>
<!ATTLIST literal-boolean
    value (true|false) #REQUIRED>

<!ELEMENT expr (%expr-elems;)>


<!ELEMENT binary-expr ((%expr-elems;),(%expr-elems;))>
```

```
<!ELEMENT lvalue (%primary-expr;)>


<!ELEMENT assignment-expr (lvalue,(%expr-elems;))>
<!ATTLIST assignment-expr
    op CDATA #REQUIRED>


<!ELEMENT unary-expr (%expr-elems;)>
<!ATTLIST unary-expr
                op CDATA #REQUIRED
                post (true|false) #IMPLIED>

<!ELEMENT if (test,true-case,false-case?)>

<!ELEMENT test (%expr-elems;)>
<!ELEMENT true-case (%stmt-elems;)?>
<!ELEMENT false-case (%stmt-elems;)?>

<!ELEMENT array-ref (base,offset)>

<!ELEMENT base (%expr-elems;)>
<!ELEMENT offset (%expr-elems;)>


<!ELEMENT type (type-argument*)>
<!ATTLIST type
    primitive CDATA #IMPLIED
    name CDATA #REQUIRED
    dimensions CDATA #IMPLIED
    idref IDREF #IMPLIED>

<!ELEMENT type-argument (type|wildcard)>

<!ELEMENT wildcard (bound?)>

<!ELEMENT bound (type)>
<!ATTLIST bound type (upper|lower) #REQUIRED>


<!ELEMENT type-parameters (type-parameter)+>

<!ELEMENT type-parameter (bounds?)>
<!ATTLIST type-parameter
                name CDATA #REQUIRED>

<!ELEMENT bounds (type)+>


<!ELEMENT local-variable
                (type,(staticinitializer|arrayinitializer|%expr-elems;)?)>
<!ATTLIST local-variable
        name CDATA #REQUIRED
        id ID #REQUIRED
        continued CDATA #IMPLIED
        %mod-final;>


 <!-- a method now has a block -->
<!ELEMENT method (type-parameters?,type,formal-arguments,throws*,block?)>
<!ATTLIST method
                name CDATA #REQUIRED
```

```
                    id ID #REQUIRED
                    %visibility-attribute;
                    %mod-abstract;
                    %mod-final;
                        %mod-static;
                    %mod-synchronized;
                    %mod-volatile;
                    %mod-transient;
                    %mod-native;>


<!ELEMENT formal-arguments (formal-argument)*>


        <!ELEMENT block ((%stmt-elems;)*)>

<!ELEMENT loop (init?,test?,update?,(%stmt-elems;)?)>
<!ATTLIST loop
                    kind (for|while|do) #REQUIRED>

<!ELEMENT init (local-variable|%expr-elems;)*>
<!ELEMENT update (%expr-elems;)*>


<!ELEMENT var-ref EMPTY>
<!ATTLIST var-ref
        name CDATA #REQUIRED
        idref IDREF #IMPLIED>


<!ELEMENT break EMPTY>
<!ATTLIST break
                    targetname CDATA #IMPLIED>


<!ELEMENT new-array (type,dim-expr*,array-initializer?)>
<!ATTLIST new-array
                    dimensions CDATA REQUIRED>


<!ELEMENT case (%expr-elems;)>

<!ELEMENT default-case EMPTY>


        <!ELEMENT paren (%expr-elems;)>

<!ELEMENT array-initializer (array-initializer|%expr-elems;)*>
<!ATTLIST array-initializer
    length CDATA #REQUIRED>


        <!ELEMENT dim-expr (%expr-elems;)>


<!ELEMENT switch ((%expr-elems;),switch-block+)>

<!ELEMENT switch-block ((case|default-case)+,(%stmt-elems;)*)>
```

Bibliografía

[B1] Aplicación de Técnicas CBR a los Sistemas de Enseñanzas Interactivos. Realizado por Ignacio Iglesias Franch, Denis Jiménez Requero y Javier Luengo Requero. Dirigido por Belén Díaz Agudo. Facultad de Informática UCM.

[B2] Página Web de jCOLIBRI: `http://gaia.fdi.ucm.es/projects/jcolibri/`

[B3] Página Web de Javy2: `http://gaia.fdi.ucm.es/grupo/projects/javy/`

[B4] Russell, S.J; Norvig, P. *"Inteligencia Artificial. Un enfoque moderno"*. PEARSON EDUCACIÓN, S.A., Madrid, 2004

[B5] Página Web de Wikipedia: `http://www.wikipedia.org`

[B6] `http://www.iiia.csic.es/People/enric/AICom.html`

[B7] Aamodt A. and Plaza E.: Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches. AI Communications, vol 7 (1994).

[B8] Bello-Tomás, J.J., González-Calero, P.A., Díaz-Agudo, B.: "JColibri: an Object-Oriented Framework for Building CBR Systems"". In Funk, P., González-Calero, P.A., (Eds.): Advances in Case-Based Reasoning, Procs. of the 7th European Conference on Case-Based Reasoning, ECCBR 2004. Lecture Notes in Artificial Intelligence, 3155, Springer, 2004.

[B9] ScriptManager.pdf por Marco Antonio Gómez Martín. Documento perteneciente a la documentación de Javy2.

[B10] JavyParaDummies.pdf por Marco Antonio Gómez Martín. Documento perteneciente a la documentación de Javy2.

[B11] Recio, Juan Antonio, Díaz-Agudo, Belén, Gómez-Martín, Marco Antonio and Wiratunga, Nirmalie: "Extending jCOLIBRI for Textual CBR". Proceedings of Case-Based Reasoning Research and Development, 6th International Conference on Case-Based Reasoning, ICCBR 2005, pages 421-435, Chicago, IL, US, Springer, August 2005.

[B12] Gómez-Martín, Marco Antonio, Gómez-Martín, Pedro Pablo, and González-Calero, Pedro A.: "Aprendizaje activo en simulaciones interactivas". Proceedings del Taller Técnicas de la Inteligencia Artificial Aplicadas a la Educación at XI Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA), pp. 49-58. Santiago de Compostela, Spain, Noviembre 2005.

[B13] Gómez-Martín, Marco Antonio, Gómez-Martín, Pedro Pablo and González-Calero, Pedro A.: "Aprendizaje basado en juegos" La Revista Icono 14. Nº 4 ISSN 1697-8293, 2004

[B14] Raghavan, V. V. and Wong, S. K. M. A critical analysis of vector space model for information retrieval. Journal of the American Society for Information Science, Vol.37 (5), p. 279-87, 1986.

[B15] coyle-representing, Lorcan Coyle, Conor Hayes, Pádraig Cunningham, "Representing Cases for CBR in XML", `citeseer.ist.psu.edu/551397.html`

[B16] Janet Kolodner, "Reconstructive Memory: A Computer Model", Cognitive Science 7 (1983): 4.

[B17] Michael Lebowitz, "Memory-Based Parsing", Artificial Intelligence 21 (1983), 363-404.

[B18] Bill Mark, "Case-Based Reasoning for Autoclave Management", Proceedings of the Case-Based Reasoning Workshop (1989).

[B19] Trung Nguyen, Mary Czerwinski, and Dan Lee, "COMPAQ QuickSource: Providing the Consumer with the Power of Artificial Intelligence", in Proceedings of the Fifth Annual Conference on Innovative Applications of Artificial Intelligence (Washington, DC: AAAI Press, 1993), 142-151.

[B20] Elaine Rich, Kevin Knight, " Inteligencia Artificial ", Segunda Edición. Mc. Graw Hill.

[B21] Sitio oficial de Protégé: `http://protege.stanford.edu/`

[B21] Aditya Kalyanpur, James Hendler, Bijan Parsia. "SMORE - Semantic Markup, Ontology, and RDF Editor."

[B22] Mindswap. Pellet OWL reasoner. `http://www.mindswap.org/2003/pellet/index.shtml`

[B23] HOWTO use Jena and DIG reasoner: `http://jena.sourceforge.net/how-to/dig-reasoner.html`

[B24] Asunción Gómez, Natalia Juristo, César Montes, Juan Pazos. "Ingeniería del Conocimiento". Editorial Centro De Estudios Ramón Areces, S.A.

[B25] María Jesús Lamarca Lapuente: Ontologías. `http://www.hipertexto.info/documentos/ontologias.htm`

[B26] Althoff, Klaus-Dieter, Ralph Bergmann, and L. Karl Branting, eds. Case-Based Reasoning Research and Development: Proceedings of the Third International Conference on Case-Based Reasoning. Berlin: Springer Verlag, 1999.

[B27] Kolodner, Janet. Case-Based Reasoning. San Mateo: Morgan Kaufmann, 1993.

[B28] Riesbeck, Christopher, and Roger Schank. Inside Case-based Reasoning. Northvale, NJ: Erlbaum, 1989.

[B29] Gómez-Martín, Marco Antonio, Gómez-Martín, Pedro Pablo, Propuesta de documento de diseño de JV2M, 2006.

[B30] Página oficial de Dublin Core. `dublincore.org`

[B31] Heramienta Ontobridge. `http://gaia.fdi.ucm.es/projects/ontobridge`

[B32] Página oficial de WordNet. `wordnet.princeton.edu`

Glosario de Términos

C

- **CBR:** "Cased Based Reasoning" o razonamiento basado en casos es un modo natural en el que razonan los seres humanos, basándose en la experiencia previa para resolver un nuevo problema.
- **CCPL:** " Creative Commons Public License". Licencia bajo la que está publicado el conversor de Java a Xml de Harsh Jain.

D

- **DIG:** Razonador utilizado para comprobar la consistencia de las ontologías.
- **DOM:** "Document Object Model" Representación de elementos XML como objetos Java.
- **DTD:** "Document Type Definition" Plantilla que marca el formato de archivo de un XML.

G

- **GAIA:** Grupo de investigación de la UCM. El nombre es un acrónimo de "Group of Artificial Intelligence Applications".

I

- **IA:** Inteligencia Artificial. Rama de la informática que trata de dotar a la máquinas de comportamientos que emulen el comportamiento humano.
- **IAIC:** Inteligencia Artificial e Ingeniería del Conocimiento. Asignatura de la Facultad de Informática de la UCM que enseña temas como búsqueda con adversario, redes semánticas, sistemas de producción, o aprendizaje automático.
- **IR:** "Information Retrieval". Es la ciencia que se encarga de buscar información en documentos, buscar documentos, o información sobre los documentos.

- **ISBC:** Ingeniería de sistemas basados en conocimiento. Asignatura de la Facultad de Informática de la UCM que enseña temas como razonamiento basado en reglas, razonamiento basado en casos, aprendizaje máquina o agentes software mediante una aproximación práctica.

J

- **Jargs:** Librería java que proporciona medios para controlar el uso de atributos de entrada.
- **Java:** Lenguaje de programación de alto nivel orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990.
- **Javy2:** Sistema de enseñanza interactivo desarrollado por el grupo de investigación GAIA cuyo objetivo es enseñar la compilación de Java de una forma lúdica.
- **jCOLIBRI:** Programa desarrollado por GAIA que constituye un marco para el desarrollo de sistemas CBR.
- **JDK:** "Java Development Kit", es un compilador y conjunto de herramientas de desarrollo para la creación de programas independientes y applets Java.
- **Jena:** Librería Java que implementa y proporciona los métodos necesarios para manejar ontologías desde Java.
- **JNI:** "Java Native Interface", JNI es un interfaz de código nativo que emplea Sun. JNI permite ejecutar código Java y comunicar con librerías escritas en otros lenguajes, como pueden ser C y C++.
- **JVM:** "Java Virtual Machine", Máquina Virtual de Java. Máquina Virtual que ejecuta la instrucciones compiladas de Java.

L

- **Log:** Mensajes de estado que proporcionan información a cerca de la traza de ejecución de una aplicación
- **Log4j:** Librería Java que proporciona los métodos necesarios para manejar la salida de los mensajes de estado producidos a lo largo de la implementación de una aplicación.

N

- **NPC:** "Non Player Character", o Personaje no jugador, en un videojuego es un personaje controlado por el ordenador.

O

- **Ontología:** Esquema conceptual de conocimiento.
- **OntoJavy:** Ontología con dominio perteneciente a los elementos de la DTD de Java utilizada para la conversión de un archivo fuente Java a un archivo XML.
- **OWL:** "Ontology Web Language" Lenguaje de marcado para publicar y compartir datos en la Web usando ontologías.

P

- **Pellet:** Razonador encargado de comprobar la taxonomía y consistencia de una ontología.
- **Protegé:** Programa informático que proporciona una interfaz para la generación, mantenimiento y gestión de una ontología.

R

- **RDF** "Resource Description Framework" Framework para metadatos en la web desarrollado por la W3C (World Wide Web Consortium).

S

- **SSII:** "Sistemas Informáticos". Asignatura de la UCM consistente en realizar un proyecto de fin de carrera.

T

- **ToDo** "To do = Para hacer" Forma de marcar una parte de código Java como tarea pendiente a realizar .

U

- **UCM:** Universidad Complutense de Madrid.
- **URI:** "Uniform Resource Identifier". Un URI es una cadena corta de caracteres que identifica unívocamente un recurso.

X

- **XML:** "Extensible Markup Language". Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium.