jCOLIBRI: Una plataforma multi-nivel para la construcción y generación de sistemas de Razonamiento Basado en Casos

TESIS DOCTORAL

Juan Antonio Recio García

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Octubre 2008

jCOLIBRI: Una plataforma multi-nivel para la construcción y generación de sistemas de Razonamiento Basado en Casos

Memoria que presenta para optar al título de Doctor en Informática

Juan Antonio Recio García

Dirigida por los doctores

Belén Díaz Agudo

Pedro A. González Calero

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Octubre 2008

para Ana

Agradecimientos

Akira Toriyama, 1986 ©Bird Studio - Planeta

Pues sí, por fin he acabado la tesis y tengo la oportunidad de dar las gracias a todos aquellos que me han ayudado durante este tiempo. He elegido esta viñeta de mi querido Toriyama porque resume perfectamente lo que he intentado que fuera este periodo: lo más importante es trabajar mucho para conseguir lo que quieres, pero no todo es trabajo, también hay que dejar tiempo para divertirse porque si no acabas volviéndote loco. Pero tampoco hay que pasarse con la diversión porque tienes que descansar para rendir en el trabajo. Y finalmente hay que comer bien para coger fuerzas -que además, está muy rico. Así que utilizaré estas cuatro "lineas maestras" para ir organizando mis agradecimientos y no dejarme nada.

Trabajar mucho. Aquí tengo que empezar por Luis Hernández ya que es la persona que me abrió las puertas de la Universidad y fue el primero en pensar que quizás yo valía para esto de la investigación (o al menos para dar clase en los CFI). Tampoco hubiera sido posible realizar esta tesis sin mis directores Belén y Pedro. Belén ha sido la perfecta organizadora que siempre me ha ayudado a saber qué tenía que hacer para sacar el trabajo adelante. Y si había grandes preguntas "filosóficas", pues para eso estaba Pedro. Realmente pienso que todo este trabajo –del que me siento tan orgulloso– se debe a su impecable dirección con la que han sabido sacar lo mejor de mí. Muchas gracias, ha sido un placer y un honor haber trabajado con vosotros.

Por último, también quiero dar las gracias a todos mis compañeros de Departamento y especialmente a la gente del 411 por todo el apoyo recibido.

Así que llegamos a la parte de *divertirse mucho*. Como no, aquí me han ayudado mis amigos, fundamentales durante estos años. Mi gran amigo Luis, con la habilidad de hacerme ver la botella medio llena y conseguir siempre una sonrisa cuando más lo necesito. Toda una década disfrutando de la mejor amistad. Guille, altruista, desinteresado, incondicional. Con la mano siempre tendida para ayudarme antes de pedirlo. Cualidad tan difícil de encontrar en un amigo y que yo tanto valoro. Mi viejo amigo Kike, con el que he vivido inolvidables experiencias, demostrando a todo el mundo lo buena gente que somos los alcarreños. Raquel y Pedro Pablo con quienes he compartido tantas cosas y disfrutado siempre de su compañía. Muchas gracias a todos, gracias por estar a mi lado en esta etapa tan importante de mi vida.

Descansar mucho. Quiero aprovechar esta parte para recordar esos buenos ratos con mi primo Javi cuando yo siempre me dormía y llegaba al cercanías por los pelos. ¡Eso sí que era duro y no la tesis! Agradecer a todos mis tíos y primos por su cariño, y dedicar este trabajo especialmente a Esther porque ella también ha superado un gran reto durante los últimos años. Todos estos agradecimientos estarían incompletos sin mandar un beso enorme a mis dos abuelas a las que quiero tanto. A decir verdad, a vosotras os dedico toda la parte de *"comer mucho"*, ya que siempre me habéis demostrado vuestro cariño en forma de suculentos platos.

Ni tanto trabajo, ni tanta diversión, ni tanto descanso hubieran sido posibles sin tener detrás a mi familia. Mamá, Papá, Mariano, todo lo que soy es reflejo vuestro y consecuencia de los valores que me habéis inculcado. La confianza, el respeto y el cariño con el que habéis apoyado incondicionalmente en todos los proyectos de mi vida hacen que esta tesis sea vuestra. Gracias por ser mi familia.

También quiero mandar un beso muy fuerte a mi "nueva" familia: José, Ana María, Laura, Rocio y Lydia. Vosotros me habéis recibido y hecho sentir siempre como uno más, lo que ha sido de gran ayuda durante este tiempo. Por eso quiero aprovechar estas líneas que quedarán guardadas en vuestra estantería para siempre y dedicaros también este trabajo.

Y quiero terminar con la persona más importante de mi vida y sin la que hubiera sido imposible que yo escribiera esto: Ana. ¿té acuerdas cuando suspendí tantas asignaturas en segundo curso y me obligaste a ir contigo a la biblioteca para estudiar? ¿De todo lo que me has ayudado durante esta etapa en la Universidad? Siempre dije que mi título de ingeniero tenían que habérnoslo expedido a medias y opino lo mismo de esta tesis.

Yo no sería lo que soy hoy sin tu amor, sonrisa, apoyo, consejo, comprensión, compañía. Tantas cualidades que resulta imposible describirlas con palabras.

Mil gracias por todo. Gracias por hacerme la persona más feliz del mundo y por disfrutar la vida conmigo. Hasta la última coma de todas estas páginas está dedicada para ti.

Resumen

El Razonamiento Basado en Casos propone la reutilización de experiencias pasadas –casos– para resolver nuevos problemas. Esta idea tan básica ha dado lugar toda una amplia familia de técnicas que permiten reutilizar la información contenida en los casos para resolver problemas de una variada naturaleza. Sin embargo, la investigación en este campo no ha venido acompañada de metodologías y plataformas que faciliten el desarrollo de aplicaciones CBR.

El objetivo de este trabajo es cubrir la carencia de una plataforma que sirva como base a la comunidad CBR para desarrollar y compartir sus aplicaciones y métodos. La plataforma presentada se denomina jCOLIBRI. Su arquitectura se divide en dos capas que cubren las distintas necesidades tanto de usuarios desarrolladores como diseñadores. Los usuarios desarrolladores son aquellos que prefieren manejar directamente el código de la aplicación, mientras que los diseñadores prefieren utilizar herramientas de composición a más alto nivel.

La parte de la plataforma orientada a usuarios desarrolladores ofrece todos los elementos básicos que sirven de base en la implementación de sistemas CBR. Además se incluyen una serie de extensiones para el desarrollo de sistemas CBR Textuales –donde los casos se encuentran en forma de textos–, sistemas CBR con conocimiento intensivo —aquellos que necesitan conocimiento complementario al de los casos–, y sistemas CBR de recomendación de productos –donde se interactúa con el usuario hasta elegir un producto o caso concreto.

La otra capa de jCOLIBRI se basa en las tecnologías de la Web Semántica para permitir la composición semiautomática de las aplicaciones CBR por parte de usuarios diseñadores. Gracias a los nuevos estándares propuestos en este área es posible representar y razonar sobre el comportamiento de los métodos CBR incluidos en la plataforma y así asistir al usuario en su composición.

Índice

Resumen **xi**

1 Introducción **1**
1.1 Estructura de la memoria 4
1.2 Publicaciones 7

2 Estado del arte **11**
2.1 Razonamiento Basado en Casos 12
2.1.1 El ciclo CBR 15
2.1.2 El conocimiento de los sistemas CBR 17
2.1.3 Procesos de un sistema CBR 19
2.1.4 Familas de sistemas CBR 22
2.1.5 Armazones CBR 28
2.2 Ontologías y la Web Semántica 30
2.2.1 El papel de las ontologías en jCOLIBRI 30
2.2.2 Las ontologías 32
2.2.3 Los lenguajes y tecnologías de la Web Semántica . . . 34
2.2.4 Las Lógicas Descriptivas 36
2.3 Arquitecturas y plataformas software 41
2.3.1 Métodos de Resolución de Problemas 41
2.3.2 Los Servicios Web Semánticos 46
2.3.3 La Plataforma J2EE 53
2.4 Diseño de Armazones 56
2.4.1 Características de los Armazones 56
2.4.2 Tipos de Armazones 60
2.4.3 Armazones CBR 62
2.5 Conclusiones 63

3 jCOLIBRI **67**
3.1 jCOLIBRI 1 68
3.1.1 Diseño y arquitectura 68

3.1.2 Funcionalidad y Características 77
3.1.3 Problemas de la plataforma 86
3.2 jCOLIBRI 2 92
3.2.1 Diseño y Arquitectura 93
3.2.2 Funcionalidad 100
3.2.3 Documentación 105
3.3 Conclusiones 108

4 CBR Textual en jCOLIBRI 111
4.1 CBR Textual Semántico 113
4.1.1 Características de un armazón genérico de TCBR . . . 113
4.1.2 El modelo teórico de Lenz para TCBR 115
4.1.3 NLP mediante Máxima Entropía 117
4.1.4 NLP mediante GATE 118
4.1.5 Representación de textos en jCOLIBRI 119
4.1.6 Métodos TCBR implementados 120
4.1.7 Medidas de similitud entre textos 122
4.2 CBR Textual Estadístico 123
4.2.1 Clustering 125
4.3 Procesamiento de consultas en lenguaje natural 126
4.3.1 Evaluación 128
4.4 Resultados Experimentales 131
4.4.1 Recuperación semántica vs. sintáctica 132
4.4.2 Recuperación y adaptación estadística 134
4.5 Conclusiones 139

5 KI-CBR mediante ontologías 141
5.1 El papel de las ontologías en jCOLIBRI 143
5.2 OntoBridge 145
5.3 Persistencia y representación de los casos mediante ontologías 147
5.4 Mecansimos de recuperación con ontologías 151
5.4.1 Similitud por contenidos 153
5.4.2 Similitud por posición 154
5.5 Las ontologías como el vocabulario del sistema CBR 157
5.6 Adaptación basada en ontologías 160
5.6.1 Adaptación Transformacional 161
5.6.2 Adaptación Constructiva 166
5.6.3 Comparación de las técnicas de adaptación 167
5.7 Autoría de casos representados mediante ontologías 169
5.7.1 Anotación mediante métodos TCBR y Análisis Formal de Conceptos 171

5.7.2 Resultados experimentales 175
5.8 Conclusiones 181

6 Diseño de sistemas CBR basado en plantillas 185
6.1 Proceso de diseño de sistemas CBR mediante plantillas 186
6.2 La librería de plantillas 188
6.2.1 Plantillas de sistemas de recomendación 190
6.2.2 Métodos para instanciar las plantillas de recomendadores 199
6.3 Representación semántica de plantillas y métodos 204
6.3.1 Representación de plantillas 207
6.3.2 Representación de los métodos 211
6.3.3 El enlace entre las dos capas de jCOLIBRI 2 219
6.4 Recuperación de plantillas 224
6.4.1 Recomendación de plantillas basada en casos 225
6.4.2 Recuperar probando 226
6.4.3 Cálculo de la similitud para la recomendación de plantillas 229
6.4.4 Ejemplo de recuperación de plantillas 234
6.4.5 Evaluación y ajuste de parámetros 236
6.5 Adaptación de plantillas 238
6.5.1 Composición de métodos 241
6.5.2 Comprobación semántica de la composición de métodos 244
6.5.3 Generación de la aplicación 246
6.6 Conclusiones 250

7 Evaluación de jCOLIBRI 253
7.1 Repercusión internacional de jCOLIBRI 253
7.2 Evaluación por parte de usuarios 257
7.2.1 Resultados de las encuestas 259
7.3 Conclusiones 263

8 Conclusiones y Trabajo Futuro 265
8.1 Conclusiones 265
8.1.1 Repercusión del trabajo de tesis 270
8.2 Aportaciones 270
8.3 Limitaciones 272
8.4 Trabajo futuro 274

A Encuestas 277
A.1 Encuesta Inicial 277
A.2 Encuesta Intermedia 285
A.3 Encuesta Final 288

B CBROnto **299**
B.1 Representación de las plantillas 300
B.2 Ejemplo de representación de una plantilla 304
B.3 Representación de los métodos 309
B.4 Ejemplo de representación de un método 312
B.5 Jerarquía de tipos 314

Bibliografía **319**

Lista de Figuras

2.1 Suposición básica del CBR 15
2.2 Ciclo CBR clásico 16
2.3 Captura de MyCBR 29
2.4 Constructores y asertos en DL 39
2.5 Correspondencia entre OWL-DL y las Lógicas Descriptivas . . 40
2.6 Componentes de un PSM y su contexto de aplicación 43
2.7 Descripción de los Perfiles en OWL-S 48
2.8 Descripción de los Modelos de Proceso en OWL-S 50
2.9 Plataforma J2EE 55

3.1 Arquitectura global de jCOLIBRI 1 69
3.2 Estructura de tareas de CBROnto 71
3.3 El papel de los conectores 75
3.4 Estructura de los casos en jCOLIBRI 1 77
3.5 División en tareas de jCOLIBRI 1 80
3.6 GUI de configuración de los conectores 81
3.7 GUI de configuración de la estructura del caso 82
3.8 Aplicación CBR independiente generada por jCOLIBRI 1 . . . 84
3.9 GUI de evaluación de sistemas CBR 85
3.10 Instanciación de métodos en jCOLIBRI 1 89
3.11 Arquitectura de jCOLIBRI 2 94
3.12 Estructura de los casos en jCOLIBRI 2 97
3.13 Organización de la funcionalidad de jCOLIBRI 2 101
3.14 Visualización de la base de casos. 105
3.15 Indice de los ejemplos de jCOLIBRI 2 106
3.16 Herramienta de acceso a los ejemplos 107

4.1 Representación de casos para CBR textual 114
4.2 Representación completa de los textos 119
4.3 Interfaz de consultas en lenguaje natural 129
4.4 Recomendador de restaurantes 133

4.5 Resultados del experimento de restaurantes 135
4.6 Recuperación en la aplicación Challenger 136
4.7 Adaptación en la aplicación Challenger 137

5.1 Captura del recomendador "Entrenador Personal" 145
5.2 Esquma de persistencia mixta en ontologías 148
5.3 Esquema de persistencia completa en ontologías 149
5.4 Ejemplo de base de casos almacenada completamente en una ontología 151
5.5 Ejemplo de medidas de similitud basadas en ontologías 157
5.6 Ejemplo de regla de adaptación en ontologías 163
5.7 Ejemplo de cuento representado con ontología 165
5.8 Comparación de adaptación constructiva y transformacional . 167
5.9 Distribución de cuentos generados mediante CBR 168
5.10 Ejemplo de FCA 173
5.11 Proceso de anotación semántica de casos 174
5.12 Anotación completa de la estructura del caso 175
5.13 Resultados del FCA 177
5.14 Exhaustividad del método de anotación semántica 178
5.15 Precisión del método de anotación semántica 179

6.1 Ejemplo de organización de la librería de plantillas 189
6.2 Sistemas de recomendación única (Single-Shot) 191
6.3 Sistemas Conversacionales tipo A 193
6.4 Sistemas Conversacionales tipo B 194
6.5 Descomposición inicial de las tareas de adquisición preferencias inicial e iterativa 195
6.6 Descomposición detallada de las tareas de adquisición preferencias inicial e iterativa 196
6.7 Descomposición de la tarea genérica de recuperación. 197
6.8 Panorámica general de CBROnto en jCOLIBRI 2 205
6.9 Editor de plantillas de jCOLIBRI 2 208
6.10 Código OWL generado por el editor de plantillas 210
6.11 Esquema de la representación OWL de los métodos de jCOLIBRI2 217
6.12 Representación OWL del método NNScoring 218
6.13 Código OWL representando un método de jCOLIBRI2 223
6.14 Captura de la herramienta de recuperación de plantillas . . . 228
6.15 Organización jerárquica de los métodos en CBROnto 232
6.16 Clasificación de los sistemas de acuerdo a características globales 233
6.17 Descripción de las plantillas durante la recuperación 234

6.18 Comparación empírica de las distintas medidas de similitud entre plantillas 237
6.19 Combinaciones óptimas de los pesos para las medidas de similitud entre plantillas 238
6.20 Vista genérica de la herramienta de adaptación de plantillas . 239
6.21 Proceso de adaptación de plantillas 242
6.22 Elección del método para resolver una tarea 243
6.23 Jerarquía de tipos en CBROnto 244
6.24 Ayuda al usuario en la asignación de variables a los parámetros 245
6.25 Ayuda al usuario en la creación de variables 246
6.26 Diseño correcto de un sistema Single-Shot 247

7.1 Gráfica de descargas de jCOLIBRI 254
7.2 Gráfica de descargas de OntoBridge 255
7.3 Gráfica de usuarios y visitas a la página web de jCOLIBRI . . 256
7.4 Mapa con la distribución de visitas a la página web de jCOLIBRI 257
7.5 Distribución de visitas a la página web de jCOLIBRI 258
7.6 Resumen de resultados de la encuesta inicial 260
7.7 Resumen de resultados de la encuesta intermedia 261
7.8 Valores máximos asignados a las plantillas 261
7.9 Resumen de resultados de la encuesta final 263

Índice de Tablas

4.1 Ejemplo de Máxima Entropía 118
4.2 Métodos de la extensión de CBR Textual 121
4.3 Ejemplo de consulta estructurada en el recomendador de restaurantes 134

6.1 Ejemplo de aplicación de los métodos de navegación mediante propuestas. 204

Índice de Listados

3.1 Definición en XML de las tareas y métodos en jCOLIBRI 1 . . 72
3.2 Ejemplo de código de caja negra 88
3.3 Ejemplo de código de caja blanca 89
3.4 Ejemplo de implementación de un componente del caso 96
3.5 Interfaz StandardCBRApplication 98
5.1 Algoritmo de la Adaptación Constructiva según Plaza y Arcos (2002) 166
6.1 Ejemplo de utilización de los métodos accesores 218
6.2 Definición de la anotación de los métodos 220
6.3 Ejemplo de método anotado 221
6.4 Definición de la anotación de los métodos accesores 222
6.5 Aplicación semicompleta generada por el editor 248
B.1 Código para la representación de las plantillas 300
B.2 Ejemplo de representación de una plantilla 304
B.3 Código para la representación de los métodos 309
B.4 Ejemplo de representación de un método 312
B.5 Jerarquía de tipos básica 314

Capítulo 1

Introducción

– "Quiero construir un martillo muy grande, bonito, y que haga muchas cosas", dijo el doctorando.

– "Pero... ¿donde están los clavos?", respondieron sus directores.

El Razonamiento Basado en Casos (CBR del inglés Case-Based Reasoning) propone la reutilización de las experiencias pasadas –casos– para resolver nuevos problemas. A grandes rasgos, esta aproximación se basa en la recuperación y adaptación de las soluciones de dichas experiencias anteriores para que puedan ser aplicadas a la situación actual. Este tipo de técnicas se fundamenta en la suposición de que problemas similares tendrán soluciones similares y que los problemas tienden a recurrir. Por lo tanto, el conocimiento necesario para recuperar y adaptar las soluciones será siempre mucho menor que el necesario para construirlas desde cero. Estas premisas han dado lugar a un amplio abanico de técnicas que permiten reutilizar la información contenida en los casos para resolver problemas de una variada naturaleza: diseño, planificación, razonamiento con textos legales o médicos, sistemas de recomendación,...

Aunque se ha desarrollado un amplio abanico de técnicas CBR, numerosos investigadores del área han identificado una carencia de metodologías o plataformas que faciliten el desarrollo de las aplicaciones CBR. La existencia de este tipo de plataforma permitiría un rápido desarrollo de los sistemas CBR, y el intercambio y reutilización de código entre investigadores. Otra clara ventaja la encontraríamos en el ámbito académico donde se podría utilizar como entorno de aprendizaje para los estudiantes de CBR.

El objetivo de este trabajo de tesis consiste en cubrir esta carencia de una plataforma para el desarrollo de sistemas CBR. La plataforma desarrollada en este trabajo de tesis se denomina jCOLIBRI y propone tanto una arquitectura de referencia para la implementación de los sistemas CBR como una

serie de implementaciones por defecto de los distintos elementos de dicha arquitectura. Por lo tanto, utilizaremos el término *plataforma* para englobar la arquitectura del sistema –diseño y organización–, junto con las distintas instanciaciones de dicha arquitectura en componentes y métodos.

La principal característica de jCOLIBRI es que está orientado tanto a usuarios desarrolladores como diseñadores de sistemas CBR. Los usuarios desarrolladores tienen conocimientos técnicos de programación y desean implementar la aplicación CBR a nivel de código. Sin embargo, los usuarios diseñadores prefieren generar las aplicaciones CBR a través de herramientas de composición de más alto nivel.

Para satisfacer las necesidades de ambos tipos de usuarios, jCOLIBRI define una arquitectura multi-nivel con dos capas, cada una dirigida a uno de estos perfiles de usuario.

La capa orientada a usuarios desarrolladores ofrece la arquitectura necesaria para integrar una gran variedad de técnicas CBR. Dentro del CBR han surgido distintas familias de técnicas con problemas específicos que requieren métodos específicos. Por lo tanto, el objetivo principal de esta capa es ofrecer la plataforma necesaria para albergar e integrar todo tipo de métodos CBR. Esta parte de jCOLIBRI puede definirse como un *armazón* porque proporciona los pilares básicos que reutilizará el usuario desarrollador para *construir* sistemas CBR. Para demostrar las capacidades de integración de esta capa de jCOLIBRI se han incluido métodos que permiten implementar sistemas CBR pertenecientes a familias completamente dispares: *sistemas de CBR textual* que razonan con casos representados como textos, *sistemas de conocimiento intensivo* que complementan la información contenida en los casos con conocimiento adicional, y *sistemas CBR de recomendación de productos* donde se plantean una serie de interacciones con el usuario para ayudarle en la selección de un determinado artículo. Además se ofrece la funcionalidad necesaria para evaluar, mantener e incluso visualizar los sistemas desarrollados.

La capa de jCOLIBRI orientada a usuarios diseñadores incluye una serie de herramientas que permiten automatizar parcialmente la *generación* de sistemas CBR mediante la composición de los elementos del armazón de la capa para desarrolladores. Estas herramientas permiten diseñar los sistemas CBR de forma gráfica y guiada, sin necesidad de conocer los detalles del código. El proceso de composición se realiza mediante *plantillas* que abstraen el comportamiento de los sistemas CBR y que pueden ser instanciadas con los diferentes componentes de la capa de desarrolladores. Este *diseño basado en plantillas* es semiautomático ya que las herramientas de jCOLIBRI utilizan información semántica de las plantillas y componentes para guiar al usuario. La información semántica se representa en torno a una ontología denominada *CBROnto* que organiza y alberga el conocimiento necesario para guiar el

diseño de sistemas CBR. En este caso, la capa se ejemplifica en el dominio de los sistemas de recomendación.

jCOLIBRI es una plataforma donde la reutilización del conocimiento es un elemento fundamental. Esta necesidad la encontramos tanto en las herramientas de composición para usuarios diseñadores como en los métodos que posibilitan el desarrollo de sistemas con conocimiento intensivo. Como ya hemos visto en el caso de CBROnto, este trabajo propone la utilización de *ontologías* para permitir la reutilización del conocimiento. Este mecanismo de representación y razonamiento proviene del área de la *Web Semántica*. Este área de investigación ha invertido un gran esfuerzo en el desarrollo de mecanismos de representación del conocimiento que permitan su reutilización entre aplicaciones. Es esta capacidad de reutilización la que convierte a las ontologías en un elemento fundamental para una plataforma como jCOLIBRI.

Las ontologías no sólo permiten representar conocimiento sobre un dominio sino que pueden utilizarse para representar el funcionamiento de componentes software. Este campo de investigación se engloba dentro de los *Servicios Web Semánticos* y nos permite razonar sobre el comportamiento de cierto método –o servicio. Por ello, las herramientas de composición de jCOLIBRI utilizarán también este tipo de estándares para facilitar y asistir al usuario en el diseño de sistemas CBR.

jCOLIBRI es una plataforma que ha evolucionado durante el trabajo de tesis. En nuestra aproximación inicial –jCOLIBRI 1– se proponía un mecanismo de diseño de sistemas CBR basado en la descomposición en tareas y la asignación de métodos a las tareas que aportaran la funcionalidad necesaria. Aunque esta versión tuvo bastante éxito, presentaba una serie de carencias que limitaban su proyección futura. Entre ellas destacaba el acoplamiento entre las herramientas de composición visuales y el código de los distintos componentes del armazón. Esto complicaba la extensión del sistema y el desarrollo de nuevos componentes. Teniendo en cuenta las dificultades encontradas en la primera versión se decidió diseñar una nueva arquitectura para jCOLIBRI 2 donde se dividía claramente la parte de la plataforma orientada a desarrolladores de la parte orientada a diseñadores. Así se llegó a la plataforma actual donde se dispone de un completo armazón para el desarrollo de sistemas CBR reutilizando el código directamente, y un conjunto de herramientas que permiten la composición semiautomática de sistemas CBR. La división en dos capas de la arquitectura hace que las herramientas de composición estén completamente desacopladas del código utilizado por los usuarios desarrolladores. De cualquier modo, aunque la arquitectura de jCOLIBRI 2 presenta numerosas mejoras, también conserva muchos elementos de su predecesora que demostraron ser muy buenas ideas de diseño.

Una vez expuestas las principales ideas del trabajo de tesis, a continuación se enumeran los principales objetivos de la misma:

- Diseñar una arquitectura de referencia para el desarrollo de sistemas CBR.
- Implementar dicha arquitectura en una plataforma que sirva como marco de referencia a la comunidad investigadora y educativa.
- Estudiar las posibles aportaciones del área de la Web Semántica para la reutilización del conocimiento entre sistemas CBR y para el desarrollo de las herramientas de composición de dicha plataforma.
- Ofrecer un armazón de bajo nivel que permita a los usuarios desarrolladores construir aplicaciones CBR mediante la extensión a nivel de código de las clases de dicho armazón.
- Demostrar la viabilidad del armazón de bajo nivel probando su capacidad para integrar diversas técnicas CBR: genéricas, CBR Textual, CBR con conocimiento intensivo, sistemas CBR de recomendación, etc.
- Desarrollar herramientas de alto nivel que permitan la composición de sistemas CBR a usuarios diseñadores que no deseen entrar en los detalles del código.
- Comprobar la viabilidad de las herramientas de composición de alto nivel.

1.1 Estructura de la memoria

Esta memoria de tesis está organizada para exponer de forma incremental los distintos elementos que componen la plataforma. Primero se describe la arquitectura global de jCOLIBRI para después detallar las características y técnicas del el armazón de la capa orientada a desarrolladores. Una vez que se han expuesto los elementos básicos necesarios para la construcción de sistemas CBR, se describen las herramientas de la capa de diseñadores que permiten su composición semiautomática. Finalmente se realiza una evaluación de la plataforma. Tomando como base esta línea argumental, los distintos capítulos de la memoria son:

Capítulo 2. Estado del arte. Este capítulo presenta los cuatro pilares fundamentales sobre los que se basa el trabajo de tesis. Primero se realiza una profunda descripción del CBR y las distintas familias de sistemas que han surgido dentro de este área. A continuación, se introducen las tecnologías de la Web Semántica que se utilizarán dentro de la plataforma y que se irán aplicando en los capítulos posteriores. En tercer

lugar se presentan las diferentes arquitecturas y plataformas software que se han influido y servido de referencia en el diseño y desarrollo de jCOLIBRI. Por último, se realiza un estudio sobre las características de los armazones destacando sus posibilidades y problemas potenciales.

Capítulo 3. jCOLIBRI. En este capítulo se presenta la plataforma jCOLIBRI. Primero se describe la primera versión del sistema junto con las carencias que motivaron el desarrollo de jCOLIBRI 2. Este capítulo describe la arquitectura global del armazón y la funcionalidad incluida por defecto en el mismo. Por lo tanto se ofrece una visión completa de toda la plataforma a excepción de los métodos y herramientas que se detallan en profundidad en capítulos posteriores.

Capítulo 4. CBR Textual en jCOLIBRI. Este capítulo describe la extensión de CBR Textual incluida en el armazón. Esta extensión ofrece una serie de métodos independientes del dominio que permiten aplicar las técnicas CBR sobre textos. Para ello se presentan dos aproximaciones distintas. La primera consiste en obtener una representación estructurada del texto y así aplicar los métodos CBR normales, mientras que la segunda permite razonar sobre la información contenida en el texto sin necesidad de dicha estructuración mediante técnicas básicas de recuperación de información. Por último, se presentan también una serie de técnicas que permitirán procesar consultas del usuario en lenguaje natural.

Capítulo 5. KI-CBR mediante ontologías. Los sistemas CBR con conocimiento intensivo (KI-CBR del inglés Knowledge Intensive) complementan la información de los casos con conocimiento adicional externo. Esto implica generalmente un problema a la hora de reutilizar dicho conocimiento en otros sistemas ya que suele incorporarse de forma específica en cada aplicación. La solución propuesta en este capítulo consiste en la utilización de ontologías para representar el conocimiento adicional ya que pueden ser fácilmente integradas a la hora de desarrollar nuevas aplicaciones CBR aunque dichas ontologías fueran diseñadas con distinto propósito.

Capítulo 6. Composición de sistemas CBR basada en plantillas. Una vez descritos los diferentes componentes del armazón, este capítulo presenta la herramienta de composición desarrollada en jCOLIBRI 2 que permite diseñar y generar de forma semiautomática los sistemas CBR. Para demostrar las posibilidades de esta aproximación se utiliza el dominio de los sistemas de recomendación. Por ello, inicialmente se presentan un amplio abanico de métodos para este tipo de sistemas CBR que también han sido incorporados al armazón. A continuación se describe la idea de utilizar plantillas que abstraigan el comporta-

miento de los sistemas CBR y que puedan ser instanciadas mediante los distintos métodos de jCOLIBRI en sistemas reales. Para ello, será necesario contar con información semántica tanto de las plantillas como de los métodos, por lo que se adaptarán las tecnologías y estándares de los Servicios Web Semánticos.

Capítulo 7. Evaluación. En este capítulo se presenta una evaluación del sistema desde dos puntos de vista distintos. El primero es su repercusión en la comunidad CBR, medida según el número de universidades y organismos de investigación que utilizan actualmente jCOLIBRI como herramienta de desarrollo de prototipos, base de proyectos de investigación o como entorno de aprendizaje del CBR. El segundo tipo de evaluación es un experimento realizado con usuarios que utilizaron las herramientas de diseño de jCOLIBRI 2 para implementar sistemas CBR.

1.2 Publicaciones

El trabajo descrito en esta memoria de Tesis ha generado una serie de publicaciones a lo largo de los últimos años. Aunque cada una de ellas ha sido referenciada en esta memoria en el lugar en el que se trata su contenido, aquí se enumeran, a modo de resumen:

1. B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García, y A. Sánchez, 2007. *Building CBR systems with jCOLIBRI.* Special Issue on Experimental Software and Toolkits of the Journal Science of Computer Programming, 69(1-3):68–75. ISSN 0167-6423.

2. B. Díaz-Agudo, G. Jiménez-Díaz, y J. A. Recio-García, 2008. *How to teach semantic web?: a project-based approach.* En J. Amillo, C. Laxer, E. M. Ruiz, y A. Young, editores, *ITiCSE*, páginas 98–102. ACM. ISBN 978-1-60558-078-4.

3. B. Díaz-Agudo, E. Plaza, J. A. Recio-García, y J. L. Arcos, 2008. *Noticeably new: Case reuse in originality-driven tasks.* En K.-D. Althoff, R. Bergmann, M. Minor, y A. Hanft, editores, 2008. *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings*, tomo 5239 de *Lecture Notes in Computer Science*, páginas 165–179. Springer. ISBN 978-3-540-85501-9. ISSN 0302-9743.

4. B. Díaz-Agudo, J. A. Recio-García, y P. A. González-Calero, 2007. *Natural language queries in CBR systems.* En *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), 2007, Patras, Greece, Volume 2*, páginas 468–472. IEEE Computer Society. ISSN 1082-3409.

5. P. A. González-Calero, B. Díaz-Agudo, J. A. Recio-García, y J. J. Bello-Tomás, 2006. *Authoring tools in jCOLIBRI.* Expert Update, 8(3):16–21. ISSN 1465-4091.

6. J. A. Recio-García, D. G. Bridge, B. Díaz-Agudo, y P. A. González-Calero, 2008. *CBR for CBR: A Case-Based Template Recommender System for Building Case-Based Systems.* En K.-D. Althoff, R. Bergmann, M. Minor, y A. Hanft, editores, 2008. *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings*, tomo 5239 de *Lecture Notes in Computer Science*, páginas 459–473. Springer. ISBN 978-3-540-85501-9. ISSN 0302-9743.

7. J. A. Recio-García, B. Díaz-Agudo, D. Bridge, y P. A. González-Calero, 2007. *Semantic templates for designing recommender*

systems. En M. PETRIDIS, editor, *Proceedings of the 12th UK Workshop on Case-Based Reasoning*, páginas 64–75. CMS Press, University of Greenwich. ISBN 987-1-904521-48-8.

8. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, J. M. GÓMEZ-BERBIS, y P. A. GONZÁLEZ-CALERO, 2008. *The WINGS of jCOLIBRI: A CBR architecture for semantic web services*. International Journal of Web Services Practices (IJWSP). Selected papers from the International Conference on Next Generation Web Services Practices (NWeSP'07), 3(1):36–41. ISSN 1738-6535.

9. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, M. A. GÓMEZ-MARTÍN, y N. WIRATUNGA, 2005. *Extending jCOLIBRI for textual CBR*. En H. MUÑOZ-AVILA y F. RICCI, editores, 2005. *Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23-26, 2005, Proceedings*, tomo 3620 de *Lecture Notes in Computer Science*, páginas 421–435. Springer. ISBN 3-540-28174-6.

10. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, y P. A. GONZÁLEZ-CALERO, 2005. *A distributed CBR framework through semantic web services*. En M. BRAMER, F. COENEN, y T. ALLEN, editores, *Research and Development in Intelligent Systems XXII (Proc. of the Twenty-fith SGAI Int. Conf. on Innovative Techniques and Applications of Artificial Intelligence,AI 2005)*, páginas 88–101. Springer. ISBN 1-84628-225-X.

11. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, y P. A. GONZÁLEZ-CALERO, 2007. *jCOLIBRI 2 Tutorial*. Informe Técnico IT/2007/02, Departamento de Ingeniería del Software e Inteligencia Artificial. Universidad Complutense de Madrid. ISBN 978-84-691-6204-0. Http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2/docs.html.

12. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, y P. A. GONZÁLEZ-CALERO, 2007. *Textual CBR in jCOLIBRI: From retrieval to reuse*. En D. C. WILSON y D. KHEMANI, editores, *Proceedings of the ICCBR 2007 Workshop on Textual Case-Based Reasoning: Beyond Retrieval.*, páginas 217–226. ISBN 978-1-85923-22-4.

13. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, y P. A. GONZÁLEZ-CALERO, 2008 (aceptado, pendiente de publicación). *Prototyping Recommender Systems in jCOLIBRI. Proceedings of the 2008 ACM Conference on Recommender Systems*. ACM press.

14. J. A. RECIO-GARCÍA, B. DÍAZ-AGUDO, y P. A. GONZÁLEZ-CALERO, Aceptado, pendiente de publicación. *Semantic Templates for Case-Based Reasoning Systems*. The Knowledge Engineering Review. Special

Issue on Ontologies Conceptualization and Epistemology for Software and System Engineering. ISSN 0269-8889.

15. J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, y A. Sánchez-Ruiz-Granados, 2006. *Ontology based CBR with jCOLIBRI*. En R. Ellis, T. Allen, y A. Tuson, editores, *Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, páginas 149–162. Springer, Cambridge, United Kingdom. ISBN 1-84628-665-4.

16. J. A. Recio-García, B. Díaz-Agudo, A. Sánchez, y P. A. González-Calero, 2006. *Lessons learnt in the development of a CBR framework*. En M. Petridis, editor, *Proccedings of the 11th UK Workshop on Case Based Reasoning*, páginas 60–71. CMS Press,University of Greenwich. ISBN 987-1-904521-39-6.

17. J. A. Recio-García, M. A. Gómez-Martín, B. Díaz-Agudo, y P. A. González-Calero, 2006. *Improving annotation in the semantic web and case authoring in textual CBR*. En T. R. Roth-Berghofer, M. H. Göker, y H. A. Güvenir, editores, *Advances in Case-Based Reasoning, 8th European Conference, ECCBR'06*, tomo 4106 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 226–240. Springer, Fethiye, Turkey. ISBN 978-3-540-36843-4. ISSN 0302-9743.

18. J. A. Recio-García, A. Sánchez, B. Díaz-Agudo, y P. A. González-Calero, 2005. *jCOLIBRI 1.0 in a nutshell. a software tool for designing CBR systems*. En M. Petridis, editor, *Proccedings of the 10th UK Workshop on Case Based Reasoning*, páginas 20–28. CMS Press, University of Greenwich. ISBN 1-904521-30-4.

Capítulo 2

Estado del arte

– De acuerdo. Entonces necesito un martillo, clavos y una buena caja donde organizarlo todo. ¿Cómo se hará todo eso?

En este capítulo presentaremos las cuatro áreas donde versa este trabajo de tesis doctoral. La más importante es el Razonamiento Basado en Casos (conocido comúnmente como CBR, del inglés Case-Based Reasoning) ya que el objetivo principal es el desarrollo de un armazón que permita la generación de este tipo de sistemas. Como se expondrá en la Sección 2.1, este tipo de razonamiento propone la reutilización de experiencias pasadas para la resolución de problemas. Debido a la homogeneidad de las fuentes de experiencia (conocidas como casos) y sus posibles formas de reutilización han surgido distintas subáreas especializadas dentro del CBR. Como jCOLIBRI ofrece el soporte necesario para implementar sistemas CBR de distinta naturaleza, dicha sección también incluye un breve resumen donde se exponen las características más importantes de cada familia de sistemas CBR.

Una vez realizada la introducción al campo del CBR, en la Sección 2.2 se presentan las distintas tecnologías utilizadas en el armazón para la composición de este tipo de aplicaciones. El amplio conjunto de métodos incluidos en jCOLIBRI dificulta el proceso de generación de sistemas CBR donde se combinen dichos métodos, y por ello, una parte muy importante de este trabajo de tesis se centra en el estudio de mecanismos que permitan su composición. Más concretamente, nuestro trabajo utiliza las tecnologías del área de la Web Semántica para representar el comportamiento de los métodos del armazón y así agilizar su proceso de composición. Esta representación semántica de los métodos se basa en ontologías (cuya definición se detalla en dicha sección) que permiten estructurar y razonar sobre el conocimiento. Mediante las ontologías se describirá el comportamiento de los métodos y se realizarán las inferencias necesarias para permitir su composición. Las capacidades de

razonamiento utilizadas en jCOLIBRI se basan en las Lógicas Descriptivas por lo que dicha sección también incluirá un resumen de sus características más importantes.

La representación del comportamiento de los métodos del armazón no son el único uso de las ontologías en este trabajo. Según se introducirá en la Sección 2.1 y más detalladamente en el Capítulo 5 existe un tipo de sistemas CBR donde la información contenida en los casos se complementa con conocimiento adicional que mejora el proceso de razonamiento. En nuestro trabajo hemos utilizado también las ontologías para representar este conocimiento complementario y por ello jugarán un papel muy importante a lo largo de toda la memoria de tesis.

La siguiente sección de este capítulo se centra en las arquitecturas y plataformas software utilizadas en el desarrollo del armazón. Si bien las dos primeras secciones introducían en qué consistía el trabajo a realizar, la Sección 2.3 se encarga de presentar las distintas posibilidades para desarrollar el armazón. Inicialmente se describirá la aproximación utilizada en la primera de las dos versiones de jCOLIBRI basada en los Métodos de Resolución de Problemas donde se plantea una arquitectura que separa la funcionalidad del conocimiento del dominio. A continuación se presentará la opción utiliza en la segunda versión del armazón que tiene su origen en los Servicios Web Semánticos. Esta línea de investigación surge a partir del trabajo en la Web Semántica y plantea distintas aproximaciones para la composición de servicios distribuidos a través de la Web. Al existir un claro paralelismo entre las necesidades de representación de los Servicios Web Semánticos y los métodos de jCOLIBRI expondremos las distintas arquitecturas propuestas en este campo y que han sido utilizadas como base en nuestro armazón.

Los Servicios Web Semánticos son además una evolución de los Métodos de Resolución de Problemas, por lo tanto, encajan perfectamente en el marco de este trabajo donde la Web Semántica tiene también un papel muy importante.

El cuarto y último bloque de este capítulo expone las características generales de los armazones una vez que ya se ha expuesto qué tipo de armazón se desea implementar y qué arquitecturas software se utilizarán en su diseño. Los armazones presentan unas características propias dentro de los sistemas software genéricos que deben ser tenidas en cuenta antes y durante su desarrollo según se expondrá en la Sección 2.4.

2.1 Razonamiento Basado en Casos

El Razonamiento Basado en Casos surge en la década de los 80 como respuesta a los problemas típicos de los Sistemas Basados en Conocimiento (KBS,

del inglés Knowledge Based Systems) donde el modelo del dominio estaba representado principalmente mediante reglas. En CBR, el conocimiento no se procesa y representa como reglas sino en un conjunto de *casos* que almacenan experiencias previas. Los casos representan problemas que han sido resueltos anteriormente y estas experiencias pueden reutilizarse para resolver problemas similares adaptando a la nueva situación las soluciones que fueron válidas en el pasado. De esta forma se presentaba una aproximación completamente diferente para la construcción de los Sistemas Basados en Conocimiento.

Los Sistemas Basados en Conocimiento o Sistemas Expertos representan una de las áreas más exitosas de la Inteligencia Artificial. Desde los primeros sistemas de los años 70 hasta hoy, la repercusión en la investigación y la industria de los KBS ha sido de gran importancia. El objetivo principal de este tipo de sistemas es ayudar a los seres humanos en la resolución de problemas y toma de decisiones.

La principal diferencia entre un KBS y un programa convencional radica en su estructura y organización. En los sistemas software convencionales, el conocimiento sobre el dominio se entremezcla con el flujo de control de la aplicación. Sin embargo, los Sistemas Basados en Conocimiento dividen claramente estos dos roles (Newel, 1982). Por un lado contienen una *base de conocimiento* (o conocimiento del dominio), y por otro, el *motor de inferencia* necesario para manejar dicho conocimiento. Esta separación explícita entre conocimiento y control facilita la inclusión de nuevo conocimiento y la extensión de las capacidades del sistema. Aquí existe una analogía con el proceder de los humanos, ya que el proceso de razonamiento suele permanecer invariable mientras que se añade continuamente nueva información en forma de experiencia.

La base de conocimiento es el componente principal de este tipo de sistemas y debe representar la información sobre el dominio de la aplicación. Para ello se utilizan sistemas clásicos de representación como reglas lógicas, redes semánticas, marcos, objetos, ... Sin embargo, la obtención de este conocimiento supone el principal problema de los KBS, comúnmente conocido como *el cuello de botella de la adquisición de conocimiento* (Forsythe y Buchanan, 1993). Los encargados de obtener el modelo del dominio por medio de entrevistas con los expertos se denominan ingenieros de conocimiento. Intuitivamente, esta aproximación no es válida en dominios complejos o difíciles de modelizar donde no sea fácil representar el modelo de conocimiento necesario para el razonamiento. Además, los expertos suelen ser reticentes a este tipo de procesos debido a su tediosidad y problemas de entendimiento con el ingeniero de conocimiento.

Aparte del problema principal en la adquisición del conocimiento existen otro tipo de inconvenientes (Watson y Marir, 1994). La implementación de estos sistemas es un proceso muy complejo que debe llevarse a cabo por

personas muy especializadas dentro grandes y costos proyectos. Además, una vez implementados su eficiencia suele ser baja debido a la dificultad en la gestión de grandes volúmenes de información. Por último, y no menos importante, el mantenimiento y extensión de este tipo de software suele ser muy laborioso.

Es a partir de este escenario donde surge el Razonamiento Basado en Casos como un paradigma de razonamiento que carece de dichos problemas:

- El CBR no necesita un modelo explícito del dominio ya que utiliza las experiencias pasadas (casos) en su lugar.
- El proceso de implementación se simplifica ya que la manipulación de los casos es mucho más sencilla.
- Los grandes volúmenes de información pueden ser gestionados aplicando técnicas de gestión de bases de datos.
- Los sistemas CBR pueden adquirir nuevo conocimiento fácilmente a través de nuevos casos, lo que simplifica en gran medida su mantenimiento.

Una de las características más importantes del CBR es que modela el proceso de razonamiento de una forma muy parecida al proceder de los humanos. Los seres humanos resolvemos problemas en base a nuestras experiencias pasadas y no a partir de un conocimiento profundo del dominio en cuestión ni de sus reglas de funcionamiento. Por ejemplo, cuando un médico diagnostica un paciente no busca todas las enfermedades que coinciden con alguno de los síntomas, sino que busca conjuntos de síntomas conocidos y comunes que tienden a recurrir. Esto nos lleva a dos de los principios más importantes del CBR (Kolodner, 1996): *problemas similares tienen soluciones similares* y *los problemas tienden a recurrir*. Estas suposiciones dividen el espacio de resolución de problemas en dos espacios distintos: el espacio de descripciones de los problemas y el espacio de soluciones de dichos problemas. Además, estas hipótesis afirman que si dos problemas son vecinos en el espacio de descripciones, sus soluciones también lo serán en el espacio de soluciones. Por lo tanto, si un sistema es capaz de obtener los problemas más similares a la consulta planteada por el usuario podrá encontrar su solución ya que ésta deberá ser similar a la de los problemas recuperados al comparar las descripciones. De este modo aparecen dos de los componentes más importantes en un sistema CBR aparte de los casos: el conocimiento de similitud. Este conocimiento será distinto para el espacio de descripciones y el espacio de soluciones, pero en cualquier caso será mucho menor que el necesario para resolver el problema desde cero como se hace en los Sistemas Basados en Conocimiento clásicos. Esta suposición básica sobre el funcionamiento de

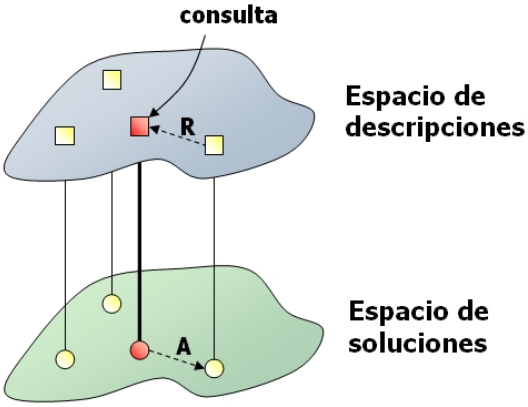


Figura 2.1: Suposición básica del CBR

los sistemas CBR se muestra gráficamente en la Figura 2.1. En dicha figura, la distancia R representa la similitud entre descripciones, mientras que la distancia A representa la similitud entre soluciones.

2.1.1 El ciclo CBR

El Razonamiento Basado en Casos es una técnica muy apropiada para dominios donde el modelo de conocimiento es difícil de obtener o representar. Los casos representan experiencias previas y así se simplifica el problema del cuello de botella presente en la mayoría de Sistemas Basados en Conocimiento ya que a los expertos les resulta mucho mas sencillo "contar batallitas" que tratar de representar el dominio mediante reglas o modelos complejos. Estas dos aproximaciones de utilizar casos o crear un modelo formal no son excluyentes ya que en numerosas ocasiones el modelo puede ser complementado con información contenida en los casos. Además, los sistemas CBR siempre van a necesitar conocimiento sobre el dominio para poder adaptar las soluciones de los casos a las nuevas situaciones. La gran ventaja del CBR es que esa información de adaptación es en principio mucho menor que el conocimiento global sobre el dominio.

La Figura 2.2 muestra el modelo clásico de razonamiento CBR propuesto por Aamodt y Plaza (1994). Este modelo divide el ciclo de razonamiento en cuatro etapas (conocidas popularmente como las 4 Rs): *Recuperación*, *Reutilización*, *Revisión* y *Retención* (del inglés Retrieve, Reuse, Revise y Retain). A grandes rasgos un sistema CBR recibe una consulta del usuario y recupera los casos más similares de la base de casos. Las soluciones de esos casos más similares se reutilizan para construir una solución al problema planteado en la consulta. Dicha solución es entonces revisada, quizás por un humano, para validarla. Finalmente la consulta junto con la solución generada es almace-

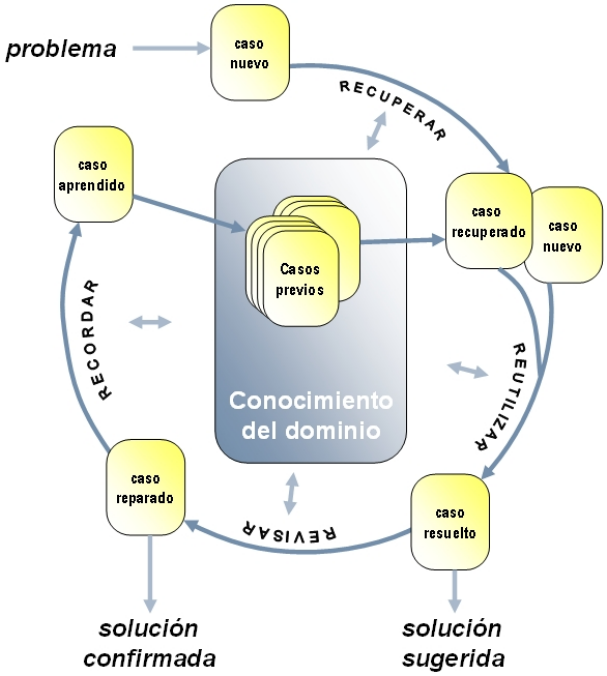


Figura 2.2: Ciclo CBR clásico

nada en la base de casos para aprender de dicha experiencia.

La característica más importante del CBR es su capacidad de aprendizaje. Esta capacidad se basa en un hecho muy sencillo: si reutilizamos las soluciones de casos similares para generar la solución de un nuevo problema, podemos almacenar dicha solución en la base de casos junto con la descripción del problema resuelto. De esta forma el conocimiento sobre la nueva situación y su solución podrán ser utilizados de nuevo para resolver futuros problemas.

La forma de solucionar problemas mediante la reutilización de experiencias pasadas tiene la ventaja inherente de aumentar la eficiencia del proceso de razonamiento. Al adaptar una solución anterior en vez de construirla desde cero se simplifica enormemente la complejidad de dicho proceso ya que no es necesario aplicarlo completamente de nuevo. En lugar de ejecutar todo el proceso necesario para alcanzar la solución sólo se requiere ejecutar un mecanismo de adaptación de soluciones. Además, el CBR permite almacenar "información negativa" sobre soluciones fallidas. Ésto evita la repetición de soluciones erróneas que no funcionaron anteriormente. También es posible guardar junto a la solución ciertas explicaciones sobre el proceso de razonamiento de cada solución y así poder explicar al usuario por qué se propone esa forma de resolver el problema. De esta manera se aumenta la acepta-

ción por parte del usuario de las respuestas del sistema ya que es posible "explicar" cómo se obtuvieron.

2.1.2 El conocimiento de los sistemas CBR

El pilar fundamental de un sistema CBR son sus casos. Citando a Leake (1996) podemos definir un caso como:

> "Un fragmento contextualizado de conocimiento que representa una experiencia que enseña una lección importante para conseguir los objetivos del razonador."

Esta breve frase describe dos características muy importantes de los casos. Para empezar, los casos son útiles en contextos concretos y para ello es necesario identificar las características que mejor describen cada contexto. Además, un caso contiene una experiencia que enseña algo, por lo que no será útil si no aporta conocimiento al proceso de razonamiento.

En general podemos organizar los casos en cuatro partes:

- La *descripción* representa el problema a resolver, situación a interpretar, artefacto a diseñar, etc. Esta parte codifica el estado del mundo a partir del cual se realiza el razonamiento. Puede contener objetivos, restricciones sobre los objetivos y cualquier otro tipo de información que limite el contexto al que que es aplicable la solución del caso. Esta parte del caso es la que se utiliza comúnmente para representar la consulta del usuario. Por lo tanto, podríamos decir que una consulta a un sistema CBR es un caso normal pero sin los elementos que describimos a continuación.

- La *solución* contiene la información necesaria para alcanzar los objetivos que aparecen en la descripción. También pueden tener formas muy diversas: planes, diseños, textos,...

- El *resultado* de aplicar la solución. Teniendo en cuenta que una determinada solución puede fallar, interesa guardar junto con la solución el resultado de aplicarla al mundo real. De esta forma el sistema CBR podrá aprender sobre soluciones fallidas y dejar de proponerlas en futuras situaciones.

- La *justificación* del caso contiene trazas que indican cómo se ha llegado a las solución y que ayudan a explicar el proceso de razonamiento al usuario. Esta última parte no es tan frecuente como las tres anteriores.

Aunque la mayoría de sistemas CBR suelen coincidir en la forma de estructurar los casos, no existe un consenso claro sobre qué información incluir en ellos. Sin embargo, hay dos consideraciones pragmáticas (Watson, 1997) que

pueden ser tomadas en cuenta al decidir la representación de los casos: la funcionalidad de la información y su facilidad de adquisición.

Además esta información puede variar dependiendo el tipo de aplicación CBR. Algunos sistemas incluyen simplemente valores numéricos, simbólicos, caracteres,... Sin embargo otras aplicaciones necesitarán almacenar textos, referencias a posiciones en taxonomías, diseños complejos o cualquier otro tipo de información que pueda ser utilizada por los métodos aplicados a lo largo del ciclo CBR.

Las formas de representar los cuatro elementos del caso descritos anteriormente pueden variar según el dominio de la aplicación. En Althoff et al. (1995) se distinguen dos grandes grupos de representaciones para los casos.

- Representaciones planas. En este tipo de representación se define una serie de atributos, cada uno de ellos con un conjunto de posibles valores de tipos simples –números, símbolos, caracteres,etc. En general no existe ninguna relación entre los atributos ni sus valores. Los casos se definen únicamente como un conjunto de tuplas atributo-valor.

- Representaciones estructuradas. También utilizan listas de atributos y valores asociados, pero a diferencia de las representaciones planas, los valores pueden ser objetos que a su vez tienen atributos. De esta forma se consiguen descripciones más expresivas en las que se definen relaciones entre los atributos y/o entre los valores. Para implementar este tipo de representaciones pueden utilizarse cálculo de predicados, redes semánticas, lenguajes basados en marcos, lenguajes orientados a objetos, lógicas descriptivas, ...

  En Bergmann y Stahl (1998) se presenta un esquema general para la representación estructurada de casos basada en objetos. Los casos estarían representados como objetos que son instancias de alguna clase que determina su estructura, es decir, sus atributos y tipos de valor de su relleno. Además, típicamente las clases que definen los objetos se presentan en una jerarquía con herencia. Respecto a los atributos se distinguen dos tipos. Los atributos *simples* son aquellos cuyos rellenos son valores de alguno de los tipos predefinidos, por ejemplo números o símbolos. Los atributos *relacionales* son aquellos cuyos rellenos son objetos de otras clases de la jerarquía, y son los que permiten representar estructuras complejas para los casos.

Por supuesto, el tipo y la estructura de representación de casos influyen en los procesos encargados de manejar dichos casos. Por ejemplo, la valoración de la similitud es muy sencilla cuando manejamos representaciones planas pero se complica según aumenta la estructuración de los casos (Matuschek y Jantke, 1997; Bridge, 1998).

El uso de representaciones estructurales presenta varias ventajas. Por un lado la expresividad, ya que ofrece una forma natural de describir objetos compuestos cuya representación mediante pares atributo-valor causaría algunos problemas típicos. Por ejemplo qué hacer con los atributos que son irrelevantes para una determinado caso. Además, la representación estructurada de casos ofrece la posibilidad de considerar las subpartes de los casos como casos en sí mismos. Estas partes pueden ser almacenadas, recuperadas y utilizadas para resolver (sub)problemas de los nuevos casos.

Como ya hemos comentado, los casos no son la única fuente de conocimiento en un sistema CBR. El origen, representación y uso de este conocimiento varía de una aplicación a otra, y puede ser utilizado tanto para calcular la similitud, adaptación o en otros procesos del ciclo CBR. Este conocimiento se representaba en la Figura 2.2 como el conocimiento del dominio que engloba a los distintos casos de la aplicación CBR.

Volviendo también a la Figura 2.1 el conocimiento de adaptación (A) y de recuperación (R) quedaría fuera del ámbito de los casos y debe ser incluido dentro del sistema CBR. La cantidad de conocimiento necesario para superar estas dos distancias dependerá de la cantidad de elementos presentes en el espacio de descripciones y soluciones. Si dichos espacios contienen una gran cantidad de casos estaremos ante una aproximación rica en datos (*Data Intensive*) donde los casos ofrecen el conocimiento necesario por A y R. Si la situación es la contraria –es decir, no existen demasiados casos– estaremos ante una aproximación más rica en conocimiento (*Knowledge Intensive*) donde el conocimiento de adaptación y recuperación debe añadirse a la aplicación CBR de forma externa a los casos.

Esta heterogeneidad en el origen y explotación del conocimiento hace difícil definir un mecanismo genérico para su inclusión en los sistemas CBR. Sin embargo, dentro del Capítulo 5 se presenta una aproximación para reutilizar el conocimiento complementario a los casos mediante ontologías.

2.1.3 Procesos de un sistema CBR

Una vez descritos los tipos de conocimientos involucrados en el Razonamiento Basado en Casos queda detallar los distintos procesos que utilizan dicho conocimiento.

Antes de empezar a explicarlos hay que destacar los distintos mecanismos para organizar los casos y facilitar su acceso. Una buena organización en memoria de los casos permitirá una recuperación más eficaz y conseguirá que los casos similares se localicen más eficientemente. Este problema de la indexación de la memoria de casos puede dividirse en dos etapas: determinar los índices de los casos y organizar los casos de acuerdo con dichos índices.

En los dos tipos de representaciones descritas (plana y estructurada) se

utiliza un conjunto de atributos para describir el caso. Entre estos atributos hay que decidir cuales son útiles para localizar casos relevantes. Los atributos que no son válidos para indexar los casos proporcionan información contextual pero no se utilizan en la recuperación. Por ejemplo, en un sistema médico identificaríamos la edad, sexo, altura y peso como atributos válidos para la indexación y recuperación de casos. Sin embargo, un atributo que contuviera la fotografía del paciente no sería válido para este proceso.

Según Lopez de Mantaras y Plaza (1997) una buena indexación no es suficiente para optimizar la recuperación sino que se necesita una organización posterior de la base de casos mediante dichos índices. La organización de los casos viene ligada normalmente con el método de similitud a utilizar. Por lo tanto, se utilizará la organización que más optimice el proceso de recuperación y el cálculo de similitud.

Una opción muy sencilla para organizar los casos consiste en utilizar una estructura plana. Este tipo de organización no utiliza los índices de los casos y requiere un recorrido lineal de la base de casos para poder computar la similitud. Mejorando esta aproximación surgen las organizaciones jerárquicas donde se utilizan los índices para agrupar los casos y permitir la selección de un subconjunto de ellos. Aunque el aumento de eficacia es significativo, a veces no es posible garantizar que el caso devuelto sea el mejor.

Una aproximación muy común dentro de las representaciones jerárquicas son los árboles de decisión. De esta forma, los valores de los índices permiten situar los casos en una parte u otra de la estructura. Entre este tipo de árboles cabe destacar los generados mediante la familia de algoritmos ID3 (Kolodner, 1993) o los árboles k-d. Esta última organización está pensada para recuperar los puntos más cercanos a uno dado, por lo que es muy apropiada para el cálculo del vecino más próximo.

Una vez que los casos han sido representados e indexados, el sistema CBR puede ejecutar los distintos procesos del ciclo CBR expuestos en la Figura 2.2: recuperación, reutilización, revisión y aprendizaje.

La recuperación es una de las partes más importantes dentro de un sistema CBR y ha sido el centro de gran parte de la investigación. En general, cuanto más eficiente sea la recuperación de casos mejor será la solución obtenida. El algoritmo de recuperación más utilizado es el de los *k vecinos más próximos* (conocido como k-NN del inglés k Nearest Neighbors). Este algoritmo compara la consulta con las descripciones de los casos y asigna un valor de similitud a cada uno. Seguidamente se recuperan los k casos con similitud más alta.

Existen diversas formas de calcular la similitud dependiendo de la representación de los casos. Normalmente los casos se representan como un conjunto de pares atributo-valor por lo que se puede obtener su similitud por medio de dos tipos de funciones. La *función de similitud local* calcula el

parecido de cada uno de los atributos, mientras que la *función de similitud global* realiza una media ponderada de los valores devueltos por las medidas locales.

Aunque la suposición de que descripciones similares conducen a mejores soluciones representada en la Figura 2.1 es la más utilizada, varios autores han señalado que esto puede modificarse en determinadas situaciones. Una opción es recuperar los casos cuyas soluciones sean más útiles en el proceso de adaptación en vez de recuperar los más similares (Bergmann et al., 2001). Otra posibilidad muy extendida en los sistemas CBR de recomendación consiste en recuperar casos similares pero diversos. Como se verá en el próximo apartado, los sistemas de recomendación son un tipo de aplicaciones CBR donde los casos son elementos o productos en vez de experiencias pasadas. En este tipo de sistemas cuando se recuperan los casos más similares a la consulta suele ocurrir que dichos casos son muy parecidos entre ellos. Si el recomendador presenta dichos casos al usuario estará limitando sus opciones de elección. Por lo tanto resulta muy interesante combinar algoritmos de similitud y algoritmos de diversidad para realizar la recuperación (Smyth y McClave, 2001).

También se han presentado aproximaciones donde se tienen en cuenta los casos que mejor representan los requisitos del usuario (McSherry, 2002a) o que son más útiles para explicar el proceso de razonamiento que llevó a la solución (Cunningham et al., 2003).

Una vez realizada la recuperación se llega a la etapa de reutilización o adaptación. En este punto se construye una solución para la consulta basándose en las soluciones de los casos recuperados. Generalmente es posible catalogar las técnicas de adaptación en dos grandes grupos: *transformacional* o *derivacional* (Kolodner, 1993; Watson, 1997).

En la adaptación transformacional se aplican los operadores o métodos de adaptación directamente sobre la solución almacenada en los casos. Sin embargo, la adaptación derivacional aplica de nuevo el proceso de obtención de la solución. Para ello se utiliza información almacenada en los casos como guía del proceso de generación de la solución.

Existen numerosas técnicas de adaptación que han sido utilizadas en CBR. Estas técnicas no son excluyentes y pueden combinarse (Lopez de Mantaras y Plaza, 1997).

La fase de revisión de la solución generada es muy problemática y solamente se aplica en una minoría de sistemas CBR. La razón es que necesita una cantidad muy considerable de conocimiento sobre el dominio y un gran esfuerzo de razonamiento. Hay algunos intentos como los presentados por Leake (1992, 1996). Uno de ellos consiste en utilizar las soluciones de los otros casos para evaluar la calidad de la solución generada. Sin embargo, la

mayoría de sistemas acaban delegando esta tarea al usuario.

En general el proceso de revisión se divide en dos partes. La primera fase consiste en *evaluar* la corrección de las soluciones. Para ello se puede intentar aplicarlas en una situación real o recurrir a algún agente externo (el usuario, por ejemplo). A continuación se ejecutaría la segunda etapa de *reparación*. Aquí se utiliza información generada por la evaluación que sirve para localizar los elementos a corregir.

La fase de retención o aprendizaje se encarga de almacenar en la base de casos la solución generada junto con la descripción correspondiente.

La primera cuestión a decidir al resolver esta fase es si el caso debe ser almacenado o no. Si se almacenan todos los nuevos casos generados se puede desembocar en una degradación de la eficiencia debido a la saturación de la base de casos. Además muchos casos pueden no aportar nada nuevo a la hora de utilizar su solución para generar una nueva. Esta decisión puede tomarse de forma autónoma o delegando en el usuario.

Otro factor a tener en cuenta es la organización de la base de casos. En organizaciones simples como las listas es muy fácil añadir casos pero hay otros muchos tipos de indexaciones donde no es tan sencillo.

Aunque técnicamente está fuera del ciclo CBR, podemos añadir otra fase de mantenimiento que se encarga de gestionar los casos almacenados en el sistema. Este tipo de técnicas están muy ligadas con la fase de aprendizaje e incluyen mecanismos para averiguar qué casos son útiles para el razonamiento y cuales añaden "ruido" a este proceso. Esta cuestión se conoce como el *problema de utilidad* (Smyth y Keane, 1995).

2.1.4 Familas de sistemas CBR

Los fundamentos del CBR se basan en la intersección de una serie de diversas disciplinas que realizaron distintas aportaciones desde el punto de vista filosófico, teórico o matemático.

El campo de las ciencias cognitivas aportó los conceptos básicos de *experiencia*, *memoria* y *analogía*. El trabajo de Schank y Abelson (1977) suele citarse como el origen del CBR y se basa en estos conceptos y teorías asociadas (Tulving, 1972; Smith et al., 1978). Schank representó las experiencias como guiones prototípicos: ir a un restaurante, al médico, etc. Su trabajo continuó explorando el papel de la memoria de situaciones anteriores (casos) en la resolución de problemas y aprendizaje. A partir del modelo de memoria dinámica desarrollado por Schank, Janet Kolodner desarrolló CYRUS; la primera aplicación CBR. Este modelo basado en casos serviría como base a los

siguientes sistemas CBR: MEDIATOR (Simpson, 1985), CHEF (Hammond, 1989), PERSUADER (Sycara, 1987), CASEY (Koton, 1988), ...

A partir de estos primeros sistemas han ido surgiendo infinidad de aplicaciones CBR muy heterogéneas y difíciles de clasificar. Sin embargo, la estructura común de dichos sistemas puede capturarse fácilmente en el modelo teórico de las 4 Rs propuesto por Aamodt y Plaza (1994). Este modelo tan sencillo es posible debido a que el CBR no ha generado su propio formalismo lógico sino que ha reutilizado los ya existentes en el campo de Inteligencia Artificial y más específicamente en Sistemas Basados en Conocimiento. Por ejemplo, las técnicas de representación del conocimiento en IA han servido para estructurar los casos o el conocimiento sobre el dominio, la tecnología desarrollada para la gestión de bases de datos se han aplicado a la organización de las bases de casos, y el conocimiento necesario para el proceso de adaptación de casos se ha representado y aplicado comúnmente mediante los sistemas de reglas existentes.

Las aplicaciones del CBR se pueden clasificar en dos grandes grupos: clasificación e interpretación de situaciones; y resolución de problemas. En general las aplicaciones industriales caen todas dentro del primer grupo, donde las técnicas a utilizar están mejor definidas, la representación de los casos es más sencilla y existen herramientas comerciales que soportan el desarrollo. Los sistemas de resolución de problemas, por el contrario, son más habituales en el ámbito académico ya que necesitan representaciones más sofisticadas y más esfuerzo de adquisición y formalización de conocimiento.

En el ámbito de los sistemas CBR de clasificación e interpretación destacan los siguientes dominios de aplicación:

- Servicios de atención al cliente (help-desk). Este es el dominio donde más éxito han tenido los sistemas CBR fuera del mundo académico. El problema fundamental de los servicios de atención al cliente es básicamente económico: si el producto sobre el que se presta soporte es sofisticado, entonces es necesario que la persona que resuelve las dudas de los clientes sea un experto, y a una empresa le resulta muy costoso dedicar personal cualificado a una tarea esencialmente repetitiva –los problemas de los clientes tienden a repetirse–. Por otra parte, es muy habitual que los servicios de atención al cliente lleven un registro de las llamadas atendidas, con lo que ya tenemos los casos. La solución basada en CBR consiste en construir un base de casos con los problemas resueltos por los expertos y encargar a una persona menos cualificada la atención de los clientes. De esta forma, el experto sólo será consultado cuando un cliente plantee un problema que no haya sido resuelto previamente. Como es natural, cada vez que los expertos resuelven un nuevo problema, este pasa a formar parte de la base de casos.

- Diagnóstico y resolución de fallos. Los sistemas de diagnóstico basados en CBR se plantean como herramientas de ayuda que el experto utiliza para acelerar su trabajo y que además sirven como una "memoria corporativa" a través de la cual distintos expertos comparten el resultado de su actividad diaria.

- Predicción y valoración. En dominios difíciles de formalizar la casuística es una fuente valiosa de predicciones y, por lo tanto, resultan aptos para las aproximaciones basadas en casos. Desde el punto de vista técnico, la principal dificultad en el desarrollo de este tipo de sistemas radica en establecer cuáles son los atributos predictivos que permiten determinar la similitud entre una situación pasada y la actual.

- Enseñanza. Diversos pedagogos argumentan sobre las ventajas de la enseñanza basada en ejemplos y es por ello que se han desarrollado trabajos dentro de lo que se ha venido en denominar "enseñanza basada en casos". Un sistema de este tipo suele incluir, además de un conjunto de ejemplos con valor pedagógico, un entorno simulado donde el alumno realiza algún tipo de interacción, de forma que, en base a dicha interacción, el sistema sea capaz de sugerir ejemplos que aporten información relevante a la tarea que en ese momento se esté realizando. Este tipo de sistemas se engloban dentro de las aplicaciones de clasificación porque su efectividad depende de que el sistema sea capaz de clasificar el estado de la interacción con el alumno y recuperar casos apropiados.

- Comercio electrónico. En este dominio se han realizado algunos trabajos que sacan partido de los procesos de "recuperación aproximada" propios del CBR. En los sistemas de comercio electrónico convencionales, la interfaz de consulta consiste en un catálogo y/o un formulario de consulta a base de datos. El problema con los catálogos es que su exploración puede resultar tediosa para los usuarios. El inconveniente de las consultas a base de datos es que recuperan lo que el usuario pide y sólo eso, lo que, en muchas ocasiones, acaba siendo una lista de resultados vacía o demasiado extensa. Si se dispone de una función de similitud entre descripciones, se puede permitir al usuario que construya consultas muy detalladas sin miedo a las listas de resultados vacías: aunque el sistema no tenga ningún producto como el solicitado, podrá ofrecer productos parecidos que puedan interesar al cliente.

  Este tipo de aplicaciones suelen englobarse dentro de los llamados "sistemas CBR de recomendación". Este tipo de sistemas –perteneciente a los sistemas Conversacionales que se introducirán en la Sección 2.1.4.3– utilizan la recuperación aproximada del CBR para recomendar productos a los usuarios. De esta forma, los casos del sistema CBR no con-

tienen experiencias pasadas que deben ser reutilizadas sino productos que son propuestos al usuario.

Por lo que se refiere a los sistemas CBR de resolución de problemas, sus aplicaciones se restringen, casi exclusivamente, al ámbito académico en dos áreas clásicas de IA: planificación y diseño.

- Los sistemas de planificación basados en CBR suelen adoptar estrategias mixtas, donde es posible generar planes desde cero pero, si existe un problema ya resuelto similar al actual, se prefiere, por razones de eficiencia, modificar la solución antigua.
- Los sistemas de diseño suelen ser aún más sofisticados que los de planificación y, en general, requieren de un proceso no trivial de adquisición de conocimiento que los hace difícilmente realizables a gran escala. Cabe señalar, sin embargo, que el sistema CLAVIER (Hennessy y Hinkle, 1992), una de las primeras aplicaciones industriales del CBR, es precisamente un sistema de diseño: el diseño de la carga de un autoclave para la fabricación de piezas en aeronáutica.

La heterogeneidad de los sistemas CBR ha promovido la creación de diversas sub-disciplinas dentro del campo del CBR. Cada una de ellas presenta una serie de características distintas que complican el desarrollo de un armazón genérico para CBR como jCOLIBRI. Sin embargo, en jCOLIBRI se ha conseguido cubrir un amplio conjunto de estas aproximaciones tan diferentes que detallamos a continuación.

2.1.4.1 CBR Textual

Un conjunto de sistemas CBR muy importante engloba los relacionados con textos. Este tipo de aplicaciones CBR presenta grandes posibilidades en dominios como el médico o legal donde almacenan su información mediante texto. Además, en la gran mayoría de escenarios donde es posible aplicar CBR para mejorar los procesos de gestión de la información ésta aparece representada mediante texto. Los textos pueden ser utilizados directamente como casos si se consigue extraer la información contenida en el lenguaje natural y estructurarla adecuadamente. El conjunto de métodos desarrollados para posibilitar la utilización de texto como origen de casos se engloban en el denominado *CBR Textual* (TCBR). La investigación en este campo se centra en resolver cuatro problemas fundamentales (Weber et al., 2006): cómo calcular la similitud entre textos; cómo obtener representaciones estructuradas de los textos; cómo adaptar los casos textuales; y cómo generar automáticamente representaciones para TCBR.

El problema de calcular la similitud entre documentos textuales puede resolverse fácilmente si aplicamos las técnicas desarrolladas en el área de

Recuperación de Información (conocido comúnmente como IR, del inglés Information Retrieval). Este campo ha experimentado un enorme avance en los últimos años debido principalmente a la importancia de los buscadores web. Uno de los primeros sistemas de TCBR fue FAQ-Finder de Burke et al. (1997a), donde se recuperaban información de las FAQ de Usenet. Para ello se utilizaban técnicas de IR junto con similitud entre palabras basándose en la ontología WordNet (Miller et al., 1990). En el sistema FAllQ (Lenz y Burkhard, 1997) se explora una nueva aproximación donde se utiliza la Extracción de Información (IE, del inglés Information Extraction) para obtener entidades de información del texto y utilizarlas en el proceso de recuperación. Por último, cabe citar el trabajo de Wilson y Bradshaw (1999) donde se utiliza una representación mixta de texto y características del mismo.

Respecto a la conversión del texto a representaciones estructuradas, el trabajo de Rissland y Daniels (1996) en el sistema SPIRE utiliza consultas estructuradas cuyos posibles valores son los índices que organizan los textos. El problema de esta aproximación es que la indexación de los textos se realizaba manualmente. Una solución a este problema se propone en el sistema PRUDENTIA de Weber et al. (1998) donde se utiliza un método semiautomático basado en técnicas de procesamiento del lenguaje y conocimiento específico del dominio. Brüninghaus y Ashley (1999) utilizan clasificadores para automatizar la creación de representaciones estructuradas del texto en su sistema SMILE.

La adaptación y generación de representaciones automáticas para TCBR es un problema muy abierto en este área debido a la complejidad del lenguaje natural. Entre los esfuerzos más notables cabe citar el trabajo de Wiratunga et al. (2004) donde se presenta un método automático para extraer características del texto. Otra opción es el sistema SOPHIA de Patterson et al. (2005) donde se utilizan técnicas de clustering para relacionar casos textuales similares.

Entre los dominios más importantes, los textos legales han supuesto un campo muy activo de investigación y fueron en gran medida un dominio precursor del nacimiento del CBR. Uno de los primeros sistemas fue HYPO (Rissland et al., 1984; Ashley, 1991) que generó toda una familia de sistemas: CABARET (Rissland y Skalak, 1991), SPIRE (Daniels y Rissland, 1997), CATO (Aleven, 1997), y SMILE (Brüninghaus y Ashley, 1999).

Una parte del trabajo de esta tesis se ha basado en el desarrollo de un armazón que permita la implementación de este tipo de sistemas de TCBR. Dicho desarrollo se presenta en el Capítulo 4.

2.1.4.2 KI-CBR

La principal virtud del paradigma CBR es que minimiza el conocimiento sobre el dominio que necesita la aplicación, ya que en su lugar se utiliza la información contenida en los casos. Sin embargo, hay que tener en cuenta

que nunca se puede suprimir completamente este elemento del sistema ya que, como mínimo, se necesitará cierto conocimiento sobre el dominio para calcular la similitud o adaptar las soluciones de los casos. Dependiendo de la cantidad de conocimiento representado en el sistema podemos hablar de distintos tipos de sistemas CBR. Si tenemos muy poco (o incluso ningún) conocimiento estaríamos hablando de sistemas CBR de sólo recuperación y si nos vamos al otro extremo tendríamos las aplicaciones CBR con conocimiento intensivo, conocidas como KI-CBR del inglés Knowledge-Intensive CBR (Díaz-Agudo y González-Calero, 2000; Díaz-Agudo y González-Calero, 2001c,b; Aamodt, 2004; Gu y Aamodt, 2005). Estas aproximaciones con mucho conocimiento están justificadas por el hecho de que un sistema será más efectivo si dispone de más conocimiento con el que razonar, pero siempre requerirá de un equilibrio para no derivar en el problema del cuello de botella de los sistemas basados en conocimiento. El capítulo 5 presenta una aproximación para la generación de aplicaciones KI-CBR que evita los problemas de cuello de botella y permite una fácil incorporación y aplicación del conocimiento en aplicaciones CBR por medio de ontologías.

2.1.4.3 CBR Conversacional

Otra familia de sistemas CBR son los conversacionales (conocidos como CCBR). En el CBR conversacional se intenta interactuar con el usuario para refinar sucesivamente sus preferencias y obtener una consulta detallada de su consulta para que sea procesada en el ciclo CBR (Aha et al., 2001; McSherry, 2002b; Aha et al., 2006). Los métodos aplicados en este tipo de aplicaciones CBR son bastante parecidos a los que aparecen en los sistemas CBR de recomendación (Bridge et al., 2006). Los sistemas de recomendación también suelen utilizar esta estrategia de refinamiento para ofrecer al usuario los productos que mejor satisfagan sus preferencias.

Este tipo de aplicaciones han sido aplicadas con notable éxito en dominios como asistencia técnica, soporte a usuarios, diagnóstico de fallos o recomendación de productos (Acorn y Walden, 1992; Gupta, 1998; McSherry, 2001; Ricci y Venturini, 2002). Otras aportaciones integran el CCBR con metodologías de resolución de problemas (Muñoz-Avila et al., 1999, 2001). El CCBR ha generado diversas herramientas entre las que podemos destacar: NaCoDae (Aha et al., 2001), ExpertClerk (Shimazu, 2002), Sermo (Bridge, 2002) y COBBER (Gómez-Gauchía et al., 2005).

La investigación en CCBR se centra en investigar cómo realizar el diálogo con el usuario para obtener sus preferencias. Empezando por la representación de los casos, normalmente se utiliza una aproximación basada en pares de preguntas y respuestas. Sin embargo, otros autores han presentado representaciones basadas en taxonomías (Gupta, 2001) o redes semánticas (Gu y Aamodt, 2005). Respecto a la adquisición de casos se ha intentado automatizar el proceso en distintos trabajos (Gupta y Aha, 2004; Weber et al., 2004).

Uno de los procesos más importantes en CCBR es la selección de las preguntas presentadas al usuario. Aquí se han investigado diversas opciones como la utilización de taxonomías (Gupta et al., 2002) o algoritmos de ganancia de información (Schmitt et al., 2002).

El capítulo 6 detalla los métodos y sistemas de CCBR desarrollados durante el trabajo de tesis para permitir la implementación de recomendadores CBR.

2.1.4.4 Otros tipos de sistemas CBR

La aplicación del CBR al diseño también ha sido un área muy activa de investigación (Goel y Craw, 2006). Los procesos de diseño complejos pueden beneficiarse de las metodologías CBR ya que normalmente se utilizan los diseños anteriores para generar nuevas soluciones. El primer sistema de este tipo fue CYCLOPS (Navinchandra, 1991) y a partir de ahí surgieron todo un abanico de aplicaciones de diseño: ARCHIE (Pearce et al., 1992), IDEAL (Bhatta y Goel, 1997), Déjà Vu (Smyth y Cunningham, 1992) o COMPOSER (Purvis y Pu, 1998) son algunos de los sistemas más representativos.

Otro exitoso campo de aplicación del CBR es el diagnóstico (Göker et al., 2006). Este tipo de sistemas deben identificar el origen de un funcionamiento anormal y proponer una solución. Las aplicaciones de este tipo de CBR varían desde el soporte a usuarios y solución de problemas técnicos: SMART (Acorn y Walden, 1992), HOMER (Göker et al., 1998) o CASSIOPÉE (Heider, 1996); hasta aplicaciones de diagnóstico médico: CASEY (Koton, 1988) o BOLERO (López y Plaza, 1997).

Por último cabe citar la Planificación Basada en Casos (CBP, del inglés Case-Dased Planning) como otro subcampo del CBR (Cox et al., 2006). Aquí los casos almacenan planes que podrán ser adaptados a nuevos problemas. Para ello se necesita almacenar los estados iniciales y finales, el plan que los une y anotaciones sobre el proceso de derivación del plan. El sistema más popular en este grupo es el sistema CHEF, donde se aplica CBP para obtener recetas de cocina (Hammond, 1989).

2.1.5 Armazones CBR

Existen varios armazones para el desarrollo de aplicaciones CBR. Sin embargo, actualmente sólo encontramos tres proyectos de código libre activos (por lo tanto, descartaremos aquí las aplicaciones empresariales que no facilitan el código fuente). Estos tres proyectos son: *MyCBR* del Centro Alemán para la investigación en Inteligencia Artificial (DFKI), *UICBRF* de la Universidad de Indiana y el propio jCOLIBRI.

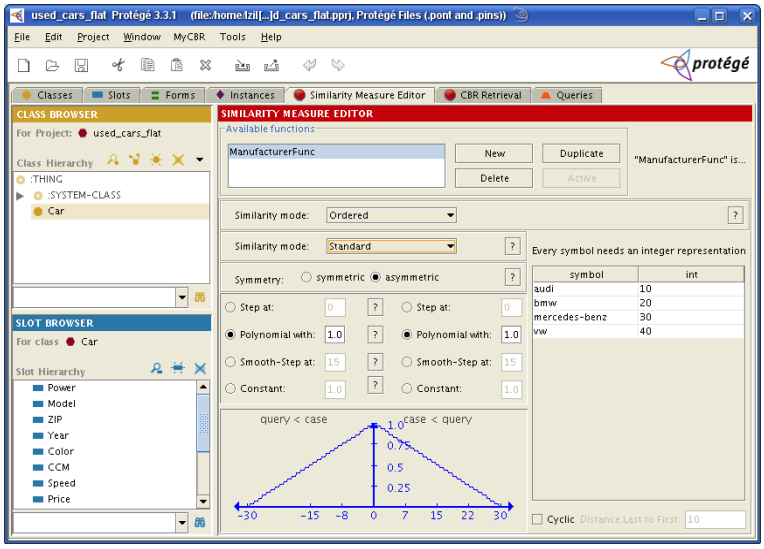


Figura 2.3: Captura de MyCBR

MyCBR (Stahl y Roth-Berghofer, 2008) es un plugin de Protege[1] que se centra en el proceso de recuperación y es especialmente útil para definir complejas medidas de similitud. Estas medidas de similitud pueden basarse tanto en las taxonomías de las ontologías o en tablas con valores de similitud. La herramienta permite definir las consultas a través de formularios y configurar la aplicación de forma gráfica. Su motor de recuperación (implementación del algoritmo k-NN) también puede utilizarse de forma independiente (sin Protege). La Figura 2.3 muestra una captura de este sistema donde se está definiendo una medida de similitud.

Por su parte *IUCBRF* (Bogaerts y Leake, 2005) es otro armazón implementado en Java que permite el desarrollo de aplicaciones CBR. Los componentes incluidos permiten implementar procesos de recuperación, adaptación, interacción (CBR conversacional), mantenimiento y monitorización de rendimiento. Los casos de este armazón se dividen en descripción del problema y solución, mientras que las organizaciones de bases de casos disponibles son: lineal y árbol-B. La utilización de los componentes de esta herramienta se realiza programando directamente el código fuente.

Además de las aplicaciones expuestas anteriormente existen otras muchas en la literatura que han dejado de desarrollarse o que han caído en el desuso. Como referencias bibliográficas podemos citar Watson (1997) donde se ofrece

[1] http://protege.stanford.edu

un resumen exhaustivo que compara software CBR, y Althoff et al. (1995) que presenta una revisión más antigua de herramientas CBR.

2.2 Ontologías y la Web Semántica

Una vez descrito el dominio de aplicación de nuestro armazón en esta sección se describen las tecnologías que permiten representar, estructurar, reutilizar y razonar sobre el conocimiento necesario en una aplicación CBR. Para alcanzar este objetivo se utilizan las tecnologías desarrolladas dentro de la Web Semántica y, más específicamente, las ontologías. Como se describirá a continuación, las ontologías permiten representar y razonar sobre el conocimiento, por lo que son muy apropiadas para describir el comportamiento semántico de un componente software, tal y como se planteará en el Capítulo 6.

Además, también utilizaremos las ontologías para encapsular y reutilizar el conocimiento del dominio complementario al de los casos que se necesita en una aplicación CBR. Como ya se comentó en la sección anterior existen dos aproximaciones distintas en el CBR dependiendo de la relación entre la cantidad de casos y el conocimiento complementario necesario. Si no existen demasiados casos es necesario añadir conocimiento que supla la falta de conocimiento declarativo de similitud y adaptación para mantener la eficiencia del sistema. Esta aproximación –denominada *Knowledge Intensive*– es la complementaria al escenario donde existen muchos casos que aportan el conocimiento declarativo anteriormente mencionado y conocido como aproximación *Data Intensive*. Esta última aproximación es bastante más sencilla que la rica en conocimiento ya que resulta difícil definir un mecanismo para representar el conocimiento necesario para complementar los casos. Sin embargo, en el Capítulo 5 presentaremos una metodología para realizar esta representación y reutilización del conocimiento mediante ontologías.

Como base a la discusión planteada en los capítulos citados anteriormente, los distintos apartados de esta sección presentan las aportaciones de esta tecnología al trabajo de tesis; una descripción detallada de las ontologías, los lenguajes y tecnologías desarrolladas dentro del área de la Web Semántica para poder utilizarlas dentro de un armazón como jCOLIBRI; y por último una pequeña introducción al lenguaje lógico que permite razonar sobre ellas: las Lógicas Descriptivas.

2.2.1 El papel de las ontologías en jCOLIBRI

Las ontologías han sido uno de los elementos principales dentro de las metodologías de desarrollo de Sistemas Basados en Conocimiento (Guarino, 1997; Crubézy y Musen, 2004; Díaz-Agudo, 2002). Las ontologías capturan el conocimiento de un dominio de modo genérico y proporcionan una comprensión comúnmente aceptada que puede ser reutilizada y compartida a través de

aplicaciones y grupos (Chandrasekaran et al., 1999; Uschold y Gruninger, 1996). En la sección anterior se ha puesto de manifiesto el problema del cuello de botella que supone la adquisición de conocimiento en KBS. Por lo tanto, siguiendo el trabajo de Díaz-Agudo (2002), en esta tesis presentaremos varias aproximaciones para utilizar ontologías como conocimiento del dominio en la aplicaciones CBR.

Esta aproximación se fundamenta en que las ontologías presentan numerosas características que las hacen muy interesantes desde el punto de vista de los Sistemas Basados en Conocimiento. En primer lugar las ontologías son representaciones de conocimiento que pueden ser intercambiadas e interrelacionadas fácilmente. Suelen estar representadas mediante lenguajes estándar que facilitan su utilización en sistemas software. Y por último, permiten el razonamiento sobre ellas por medio de razonadores que han evolucionado enormemente y presentan una gran eficiencia y estabilidad.

Desde el punto de vista concreto de los sistemas CBR podemos utilizar ontologías para almacenar o indexar los casos, o como conocimiento de similitud y adaptación. Estas aplicaciones de las ontologías para mejorar las aplicaciones CBR se explicarán el el Capítulo 5.

Sin embargo, esta no es la única aplicación de las ontologías en este trabajo. El objetivo de esta tesis ha sido crear el armazón jCOLIBRI para posibilitar el desarrollo de aplicaciones CBR. Dentro del proceso de desarrollo de sistemas CBR planteado, es necesario componer los distintos métodos incluidos en el armazón para obtener una aplicación funcional. En nuestra aproximación, dicha composición requiere conocer las capacidades, funcionamiento e invocación de los distintos componentes del armazón. Y aquí es donde también se pueden aplicar ontologías. Las ontologías permiten describir los componentes de un armazón o arquitectura de componentes y mediante el razonamiento sobre dichas ontologías es posible componerlos de forma “inteligente”.

Las ontologías han sido comúnmente utilizadas en la construcción de sistemas KBS. Sin embargo, su popularización, extensión y –aún mas importante– su estandarización se ha debido al auge de la Web Semántica. La Web Semántica es un área de investigación que se encarga de dotar de significado a la Web actual. Para ello es necesario generar una serie de estándares que permitan representar unívocamente la semántica de los distintos elementos que puedan aparecer en ella. La búsqueda de estos mecanismos de representación es la que ha promovido el desarrollo de las ontologías.

Sin embargo, la naturaleza distribuida y heterogénea de la Web ha requerido el desarrollo de estándares para la representación de dichas ontologías que permitan su utilización en cualquier tipo de plataforma o aplicación. De esta forma han surgido estándares de representación de ontologías como *OWL* fundamentados en otros ya implantados como *XML* o *RDF*. Estos

estándares se detallarán en la Sección 2.2.3.

Junto con estos lenguajes para la descripción de ontologías se han desarrollado paralelamente razonadores que permiten manejar y explotar ese conocimiento de forma eficiente. En este trabajo nosotros proponemos reutilizar todo el esfuerzo de representación y estandarización que a surgido en el ámbito de la Web Semántica para la representación del conocimiento del dominio dentro de los sistemas CBR. Concretamente nos centraremos en la representación de ontologías mediante Lógicas Descriptivas. Este formalismo ofrece un alto grado de expresividad pero que a su vez permite la implementación de razonares eficientes que trabajen sobre el mismo. Los detalles sobre este tipo de lógicas se presentarán en la Sección 2.2.4.

Como ya dijimos anteriormente, la otra aportación de las ontologías a este trabajo es la composición de sistemas CBR ya que podemos utilizarlas para describir el funcionamiento de los distintos métodos o componentes de una aplicación CBR. Esta posibilidad viene estudiándose desde hace mucho tiempo en el área de los Servicios Web Semánticos (SWS). Este subárea de la Web Semántica propone la utilización de ontologías para la descripción de los Servicios Web. Los Servicios Web son aplicaciones o métodos software que pueden ejecutarse remotamente a través de Internet. Estos servicios constituyen una de las áreas con más proyección empresarial actual ya que permiten la interacción entre agentes software para que realicen tareas complejas de forma distribuida e interrelacionando funcionalidad de distinta naturaleza. Los servicios web por sí solos no tienen demasiada utilidad ya que únicamente podrán ser utilizados por los agentes que conozcan su funcionamiento y mecanismo de invocación. Por lo tanto, se necesita dotar de significado a dichos servicios para que puedan ser descubiertos, combinados e invocados automáticamente.

Es a partir de esta búsqueda de representaciones semánticas donde el área de Servicios Web encontró en la Web Semántica los recursos necesarios para poder dotar de significado semántico al funcionamiento de los servicios y así automatizar su manejo e invocación por los agentes software. Aunque los Servicios Web Semánticos están íntimamente relacionados con la Web Semántica, serán presentados en la próxima sección desde el punto de vista de las arquitecturas software.

A continuación haremos un breve repaso a los fundamentos teóricos de las ontologías y describiremos las tecnologías desarrolladas dentro de la Web Semántica para su representación y explotación.

2.2.2 Las ontologías

La definición más extendida de ontología es la presentada por Gruber (1993):

> "Una ontología es una especificación explícita de una concep-

tualización."

Conceptualización se entiende como una estructura semántica intensional que codifica las reglas implícitas que restringen la estructura de un fragmento de la realidad. Desde un punto de vista más pragmático, podemos decir que en las ontologías se combinan varios esquemas en una estructura que contiene todas las entidades relevantes y sus relaciones dentro del dominio.

Las ontologías pueden ser utilizadas para distintas tareas dependiendo del tipo de aplicación donde se integren (van Heijst et al., 1997; Chandrasekaran et al., 1999). A continuación detallaremos una breve clasificación que en ningún modo es excluyente ya que una misma ontología puede enmarcarse dentro de varios de los siguientes tipos:

- Las ontologías de representación de conocimiento (van Heijst et al., 1997) capturan las primitivas de representación utilizadas para formalizar conocimiento en algún paradigma de representación. Estas ontologías proporcionan un marco de representación neutral respecto a las entidades concretas del mundo o del dominio que describen.

- La ontologías de conocimiento general o de sentido común capturan vocabulario relativo a cosas generales como eventos, tiempo, causalidad, espacio, comportamiento, etc.

- Las ontologías de nivel superior (Top-Level) proporcionan elementos y términos con alto nivel de abstracción, es decir, términos generales bajo los cuales se suelen colocar los términos más específicos de otras ontologías.

- Las ontologías del dominio (van Heijst et al., 1997; Riichiro et al., 1995) proporcionan vocabulario sobre los conceptos de un dominio y las relaciones entre ellos, las actividades que se desarrollan y los principios elementales que rigen el comportamiento en ese domino.

- Las ontologías lingüisticas proporcionan información sobre los elementos de un idioma. La más importante es WordNet (Miller et al., 1990) y contiene una base de datos léxica para el idioma inglés cuya información se organiza en unidades llamadas synsets o conjuntos de términos sinónimos.

- Las ontologías de tareas (Riichiro et al., 1995) proporcionan un vocabulario sistemático de los términos utilizado para resolver problemas asociados con tareas compartidas por distintos dominios. Relacionadas con éstas se encuentran las llamadas *ontologías de tareas-dominios* que son ontologías de tareas que sólo se pueden reutilizar en un cierto dominio y no en otros, debido a que la terminología asociada a la tarea está particularizada a un dominio concreto.

- Las ontologías de métodos de resolución de problemas proporcionan definiciones de los términos, conceptos y relaciones que se utilizan para especificar un proceso de razonamiento llevado a cabo para lograr una tarea (Chandrasekaran et al., 1999). Son independientes del dominio y sus elementos se describen en términos neutrales respecto a una aplicación a un dominio concreto.

- Las ontologías de aplicación (van Heijst et al., 1997) contienen el conocimiento y las definiciones necesarias para una cierta aplicación.

En este trabajo se utilizan varias ontologías de distinta naturaleza: linguísticas, de tareas, de dominio, de resolución de problemas, etc. Dichas ontologías se irán presentando en los siguientes capítulos.

El conocimiento contenido en las ontologías se formaliza por medio de cinco tipos de componentes: clases, relaciones, funciones, axiomas e instancias (Gruber, 1993). Aunque las clases de una ontología se organizan generalmente de forma taxonómica, no se debe considerar esta taxonomía como una ontología en sí misma (Studer et al., 1998).

El término concepto o clase se utiliza en un sentido amplio para agrupar conjuntos de individuos (también denominados instancias), que pueden describir a su vez tareas, funciones, acciones, estrategias, procesos, etc. Las relaciones representan un tipo de interacción entre los conceptos del dominio. Las funciones son un caso especial de relación n-aria en el que el elemento n de la relación está unívocamente determinado por los $n-1$ anteriores elementos. Los axiomas se usan para modelar sentencias que son siempre ciertas en este dominio y las instancias se usan para representar individuos concretos.

Una vez que los componentes de la ontología hayan sido identificados, la ontología puede ser implementada en algún lenguaje. En la literatura podemos encontrar numerosos lenguajes para la representación de ontologías como: CML (Schreiber et al., 1994), Ontolingua (Gruber, 1993), o LOOM (Mac Gregor y Bates, 1987). Sin embargo ninguno de ellos a obtenido la popularidad de OWL. Este es el lenguaje propuesto por el World Wide Web Consortium (W3C) para la representación de ontologías dentro de la Web Semántica y se ha convertido en un estándar completamente implantado. En la siguiente sección haremos un recorrido por la evolución de los lenguajes de la Web Semántica que desembocaron en la definición de OWL.

2.2.3 Los lenguajes y tecnologías de la Web Semántica

Desde la implantación "de facto" del lenguaje HTML como estándar para Internet, se han ido buscando lenguajes que permitieran ampliar la expresividad de los elementos que forman la Web. De esta forma se ha evolucionado hasta los lenguajes como *OWL* donde se pueden representar ontologías que

contienen información semántica. Podríamos decir que uno de los factores que ha incidido directamente en el éxito de estos lenguajes es que cada uno de ellos a ido basándose en el anterior. De esta forma se ha incrementado gradualmente la expresividad de cada uno de estos lenguajes.

El primer lenguaje en obtener una amplia aceptación fue el lenguaje *XML* donde las etiquetas no están orientadas a la presentación del documento sino a la definición de los datos contenidos. Sin embargo, el lenguaje XML únicamente permite una representación sintáctica de la información lo que supone una gran limitación. Para solventar este problema intentando representar información semántica se desarrolló el lenguaje RDF.

RDF (Resource Description Framework) es un lenguaje para representar recursos de la Web (W3C, 2004c). Generalizando la definición de recurso web, se puede decir que RDF permite estructurar información sobre cualquier elemento que pueda ser identificado en la red (incluso cuando no pueda ser accedido desde ella) mediante expresiones con la forma sujeto-predicado-objeto.

RDF describe un lenguaje para expresar relaciones simples entre recursos utilizando propiedades y valores. Sin embargo, no permite la definición de vocabularios para utilizar en sus descripciones. Estos vocabularios describen clases o tipos de recursos utilizando propiedades específicas. Por ejemplo, en la descripción de recursos bibliográficos sería deseable poder definir clases como libro, revista,... utilizando para ello propiedades como título, resumen, etc.

RDF Schema (RDFS) no define un vocabulario con diferentes clases para determinados dominios, sino que establece las facilidades necesarias para describir dichas clases junto con sus propiedades. De esta forma RDF Schema crea un sistema de tipos para RDF similar a los utilizados en los lenguajes de programación orientados a objetos donde los recursos pueden ser definidos como instancias de una o más clases. Para describir las clases se utilizan los recursos RDF Schema (W3C, 2004d): *rdfs:Class* y *rdfs:Resource*, y para las propiedades: *rdf:type* y *rdfs:subClassOf*. Por lo tanto, mediante RDFS podemos construir ontologías básicas donde aparezcan jerarquías de clases relacionadas a través de propiedades.

OWL (Lenguaje de Ontologías Web proveniente de las siglas inglesas Ontology Web Language) surge de la necesidad de representar información que necesita ser procesada por aplicaciones software en vez de ser presentada a humanos, siendo utilizado principalmente para la representación de ontologías. OWL se basa en XML, RDF y RDF-S aunque brinda más expresividad y significado semántico al definir vocabulario adicional junto con una semántica formal. Esta última característica es la que convierte a OWL en un lenguaje básico en la Web Semántica ya que permite el razonamiento sobre

los contenidos de los documentos (W3C, 2004b).

OWL define tres sublenguajes con diferente capacidad de expresividad (W3C, 2004a):

OWL Lite. Diseñado de cara a usuarios que necesitan jerarquías de clasificación básicas. Por ejemplo, soporta restricciones de cardinalidad pero sólo entre los valores 0 y 1. De esta forma, las herramientas que procesen OWL Lite pueden ser más fáciles de diseñar y proporcionar una migración rápida entre taxonomías existentes.

OWL DL. Presenta una expresividad máxima pero manteniendo la completitud computacional (todos los razonamientos erán computables) y la decidibilidad (todos los razonamientos terminan en un tiempo finito). El nombre de OWL DL se debe su correspondencia con las capacidades de razonamiento de las Lógicas Descriptivas.

OWL Full. Permite el máximo de expresividad y libertad sintáctica de RDF pero no garantiza la computabilidad de los razonamientos.

Estos tres lenguajes se engloban incrementalmente: OWL Lite $\subseteq$ OWL DL $\subseteq$ OWL Full. También añadir que OWL Full puede ser visto como una extensión de RDF, mientras OWL Lite y OWL DL pueden tomarse como una extensión de una parte de RDF. Cada documento OWL es un documento RDF, mientras todo documento RDF es únicamente un documento OWL Full, sin incluir OWL Lite ni OWL DL.

Aunque se podría pensar que RDFS es suficiente para la representación de ontologías, hay que señalar que en ese lenguaje sólo se permite la creación de subclases a través de la primitiva *subClassOf*. Sin embargo OWL introduce sus propias clases (*Class...*) y objetos (*Thing*) para diferenciar entre clases e instancias.

Podríamos decir que el gran éxito de OWL se debe a su equivalencia expresiva con las Lógicas Descriptivas (en su versión OWL-DL) lo que ha hecho posible el desarrollo de razonadores que trabajan eficientemente con las ontologías construidas mediante este lenguaje. Entre los razonadores más populares podemos citar Racer (Haarslev y Möller, 2001), Pellet (Sirin y Parsia, 2004) o FaCT (Tsarkov y Horrocks, 2006). Los procesos de razonamiento expuestos en esta memoria utilizan estos razonadores de Lógicas Descriptivas. Por ello, pasaremos a describir los fundamentos de este formalismo de representación.

2.2.4 Las Lógicas Descriptivas

Las Lógicas Descriptivas (DLs) (Baader et al., 2003) son un subconjunto "estructurado" de la lógica de primer orden computacionalmente tratable.

Podríamos clasificarlas en la familia de los marcos (*frames*) como descendientes de los lenguajes KRL y KL-ONE. La idea en estos lenguajes era definir una especificación precisa de la funcionalidad a proporcionar por una base de conocimiento y, específicamente, de las inferencias a realizar sobre esa base de conocimiento. En una base de conocimiento podemos distinguir entre *conocimiento intensional*, o conocimiento general sobre el dominio del problema, y *conocimiento extensional*, el cual es específico para un problema particular. Una base de conocimiento en DLs se compone análogamente de dos componentes: una *TBox* y una *ABox*. La TBox contiene conocimiento intensional en la forma de terminología (de aquí el nombre TBox) y se construye mediante declaraciones que describen propiedades y conceptos generales. La ABox contiene conocimiento extensional (también llamado conocimiento sobre asertos, de ahí el nombre ABox). Este conocimiento es específico para los individuos del dominio de discurso. El conocimiento intensional supuestamente no cambia con el tiempo, sin embargo el conocimiento extensional suele ser dependiente de cierto conjunto de circunstancias y por lo tanto sujeto a cambios.

El tipo básico de declaración en una TBox es la *definición de conceptos*, es decir, la definición de un nuevo concepto a partir de otros ya definidos. Por ejemplo, una mujer puede ser definida como una persona femenina escribiendo esta declaración: $Mujer \equiv Persona \cap Femenina$. Esa declaración será interpretada como una equivalencia lógica. Una posibilidad lógica que nos brinda la declaración de conceptos a partir de otros es la *clasificación*, donde se coloca un concepto en el lugar correspondiente en la jerarquía de conceptos declarada.

La ABox contiene el conocimiento extensional sobre el dominio de interés, esto es: asertos sobre los individuos. Por ejemplo, el aserto: $Femenina \cap Persona(ANA)$ define que el individuo ANA es una persona femenina. Tomando la definición anterior del concepto mujer es posible deducir que el concepto ANA es una instancia (o individuo) del concepto Mujer.

Además, es posible definir expresiones como $tieneHijo(ANA, DAVID)$ donde se especifica que ANA tiene a DAVID como hijo. Estos asertos se denominan *asertos de propiedades* o *relaciones*, mientras que los asertos del primer tipo (definición del concepto mujer) se denominan *asertos de conceptos*.

Las DLs permiten varios tipos de razonamiento mediante la TBox y la ABox. El más importante es la *subsumción*, escrito normalmente como $C \subseteq D$. Determinar la subsumción consiste en comprobar cuando el concepto denotado por D (el subsumidor) es considerado más general que el denotado por C (el subsumido). En otras palabras, la subsumción comprueba cuando un

concepto representa siempre un subconjunto del conjunto representado por el otro. Otra inferencia típica sobre conceptos es la *satisfactibilidad*, donde se comprueba que un concepto no representa un conjunto vacío. Otra capacidad de razonamiento, más relacionada con la ABox es el *reconocimiento de instancias*, donde se verifica si un individuo es instancia de (pertenece a) un concepto específico. La mayoría de tipos de razonamiento pueden ser calculados a partir del reconocimiento de instancias. Por ejemplo, la *consistencia de la base de conocimiento* puede ser calculada verificando si cada concepto admite al menos un individuo, la *realización* comprueba el concepto más específico al que pertenece un individuo, y la *recuperación* devuelve los individuos de la base de conocimiento que pertenecen a un concepto.

Una vez estudiados los fundamentos de las DLs y sus posibles aplicaciones pasaremos a detallar su sintaxis y semántica. Las descripciones elementales son los conceptos atómicos y los roles atómicos. Las definiciones más complejas pueden ser construidas inductivamente sobre ellos a partir de constructores:

$$
\begin{array}{lll}
C, D \rightarrow & A| & \text{(concepto atómico)} \\
 & \top| & \text{(concepto universal, top concept)} \\
 & \bot| & \text{(bottom concept)} \\
 & \neg A| & \text{(negación)} \\
 & C \cap D| & \text{(intersección)} \\
 & \forall R.C| & \text{(restricción de valor)} \\
 & \exists R.T| & \text{(cuantificación existencial limitada)}
\end{array}
$$

Aquí, la notación empleada define A como un concepto atómico y C, D son descripciones de conceptos. R representa un rol atómica.

La *interpretación* I de estas expresiones es un conjunto no vacío Δ^I (el dominio de la interpretación) y una función de interpretación que asigna a cada concepto atómico un conjunto $A^I \subseteq \Delta^I$ y a cada rol una relación binaria $R^I \subseteq \Delta^I \times \Delta^I$. La función de interpretación se extiende a las descripciones de conceptos a partir de las siguientes definiciones inductivas:

$$
\begin{array}{rcl}
\top^I & = & \Delta^I \\
\bot^I & = & \emptyset \\
\neg A^I & = & \Delta^I \backslash A^I \\
(C \cap D)^I & = & C^I \cap D^I \\
(\forall R.C)^I & = & \left\{ a \in \Delta^I | \forall b.(a,b) \in R^I \rightarrow b \in C^I \right\} \\
(\exists R.T)^I & = & \left\{ a \in \Delta^I | \forall b.(a,b) \in R^I \wedge b \in C^I \right\}
\end{array}
$$

Para crear conceptos complejos se utilizan constructores de conceptos. La tabla de la Figura 2.4 muestra la sintaxis y semántica de los constructores

Nombre	Sintaxis Abstracta	Semántica
Top	$\top$	Δ^I
Bottom	$\bot$	$\emptyset$
Intersección	$C \cap D$	$C^I \cap D^I$
Unión	$C \cup D$	$C^I \cup D^I$
Negación	$\neg C$	$\Delta^I \backslash C^I$
Restricción Valor	$\forall R.C$	$\left\{a \in \Delta^I \vert \forall b.(a,b) \in R^I \rightarrow b \in C^I\right\}$
Cuantif. Existencial	$\exists R.C$	$\left\{a \in \Delta^I \vert \forall b.(a,b) \in R^I \wedge b \in C^I\right\}$
Restricción	$\geq nR$	$\left\{a \in \Delta^I \vert\vert \left\{b \in \Delta^I \vert (a,b) \in R^I\right\} \vert \geq n\right\}$
numérica	$\leq nR$	$\left\{a \in \Delta^I \vert\vert \left\{b \in \Delta^I \vert (a,b) \in R^I\right\} \vert \leq n\right\}$
incualificada	$= nR$	$\left\{a \in \Delta^I \vert\vert \left\{b \in \Delta^I \vert (a,b) \in R^I\right\} \vert = n\right\}$
Restricción	$\geq nR.C$	$\left\{a \in \Delta^I \vert\vert \left\{b \in \Delta^I \vert (a,b) \in R^I \wedge b \in C^I\right\} \vert \geq n\right\}$
numérica	$\leq nR.C$	$\left\{a \in \Delta^I \vert\vert \left\{b \in \Delta^I \vert (a,b) \in R^I \wedge b \in C^I\right\} \vert \leq n\right\}$
cualificada	$= nR.C$	$\left\{a \in \Delta^I \vert\vert \left\{b \in \Delta^I \vert (a,b) \in R^I \wedge b \in C^I\right\} \vert = n\right\}$
Enlace	$R \subseteq S$	$\left\{a \in \Delta^I \vert \forall b.(a,b) \in R^I \rightarrow (a,b) \in S^I\right\}$
rol-valor	$R = S$	$\left\{a \in \Delta^I \vert \forall b.(a,b) \in R^I \leftrightarrow (a,b) \in S^I\right\}$
Definición de conceptos	$C \equiv D$	$C \equiv D$
Def. conceptos primitivos	$C \subseteq D$	$C \subseteq D$
Aserto de conceptos	$C(a)$	$a^I \in C^I$
Aserto de roles	$R(a,b)$	$(a^I, b^I) \in R^I$

Figura 2.4: Constructores y asertos en DL

más comunes. Las cuatro últimas filas muestran las primitivas de inclusión de axiomas.

Cada lenguaje de representación de DLs utiliza su propia sintaxis para representar los distintos elementos. En OWL-DL se utiliza una sintaxis basada en XML y RDFS para representar las expresiones lógicas. En la Figura 2.5 se muestran las correspondencias entre las construcciones OWL y su representación en DLs siendo C clase, P propiedad y o individuo.

OWL	DL	Ejemplo
intersectionOf	$C_1 \cap \ldots \cap C_n$	$Humano \cap Masculino$
unionOf	$C_1 \cup \ldots \cup C_n$	$Medico \cup Abogado$
complementOf	$\neg C$	$\neg Masculino$
oneOf	$\{o_1, \ldots, o_n\}$	$\{Pepe, Manolo\}$
allValuesFrom	$\forall P.C$	$\forall tieneHijo.Medico$
someValuesFrom	$\exists P.C$	$\forall tieneHijo.Abogado$
value	$\exists P.\{o\}$	$\exists ciudadanoDe.Europa$
minCardinality	$\geq nP.C$	$\geq 2tieneHijos.Medico$
maxCardinality	$\leq nP.C$	$\leq 1tieneHijos.Abogado$
SubClassOf	$C_1 \subseteq C_2$	$Humano \subseteq Animal \cap Bipedo$
EquivalentClasses	$C_1 \equiv \ldots \equiv C_n$	$Hombre \equiv Humano \cap Masculino$
SubPropertyOf	$P_1 \subseteq P_2$	$tieneHija \subseteq tieneHijos$
EquivalentProperties	$P_1 \equiv \ldots \equiv P_n$	$coste \equiv precio$
SameIndividual	$o_1 = \ldots = o_n$	$DL = Description_Logic$
DisjointClasses	$C_i \subseteq \neg C_j$	$Masculino \subseteq \neg Femenino$
DifferentIndividuals	$o_i \neq o_j$	$Pepe \neg Manolo$
inverseOf	$P_1 \equiv P_2^-$	$tieneHijos \equiv tienePadres^-$
Transitive	$P^+ \subseteq P$	$descendiente^+ \subseteq descendiente$
Symmetric	$P \equiv P^-$	$relacionadoCon \equiv relacionadoCon^-$
Class(C complete D)	$C \equiv D$	$Hombre \equiv Humano \cap Masculino$
Class(C partial D)	$C \subseteq D$	$Humano \subseteq Animal \cap Bipedo$

Figura 2.5: Correspondencia entre OWL-DL y las Lógicas Descriptivas

2.3 Arquitecturas y plataformas software

Una vez explicadas las características de las tecnologías para la representación y manejo de ontologías que utilizaremos en nuestro armazón CBR, esta sección expone las arquitecturas software que se han utilizado como referencia del diseño e implementación de jCOLIBRI.

En este punto debemos hacer una aclaración terminológica. Llamaremos *arquitectura software* al diseño de alto nivel que se basa en abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software. Y denominaremos *plataforma* a la suma de estas ideas de alto nivel más las implementaciones de referencia que permiten desarrollar el software. De esta forma, el armazón presentado en esta memoria encajaría con la última definición ya que contiene una serie de abstracciones para guiar la construcción de sistemas CBR y además provee una serie de componentes ya implementados que se utilizan para generar las aplicaciones finales.

Para el diseño lógico del armazón se han tomado como referencia diversas arquitecturas. En la primera versión del mismo se utilizó una arquitectura basada en Métodos de Resolución de Problemas (PSMs del inglés *Problem Solving Methods*), mientras que la segunda versión se basa en las ideas de diseño de los Servicios Web Semánticos (SWS). Como se expondrá en las siguientes subsecciones, podríamos decir que los Servicios Web Semánticos son una evolución natural de los PSMs donde se utilizan tecnologías más modernas pero manteniendo la misma esencia en cuanto a la arquitectura. Por lo tanto las ideas de la arquitectura de SWS son una elección muy apropiada para diseñar jCOLIBRI 2.

Aunque ambas versiones del armazón siguen la misma arquitectura, en la primera versión no se utiliza ninguna plataforma de referencia para su implementación. Sin embargo, la segunda versión del armazón toma varios conceptos y tecnologías de la plataforma J2EE. Esta plataforma define una serie de estándares para desarrollar aplicaciones empresariales multicapa y se basa en una serie de componentes que permiten gestionar varios aspectos de la aplicación automáticamente, sin necesidad de escribir código complejo[2]. La descripción de esta plataforma se incluye en la Sección 2.3.3 junto con las razones de su adecuación para el desarrollo de jCOLIBRI.

2.3.1 Métodos de Resolución de Problemas

Los Métodos de Resolución de Problemas surgen como una respuesta a los problemas de reutilización de software en los Sistemas Basados en Conocimiento. Como ya se comentó en la Sección 2.1 este tipo de aplicaciones suelen ser bastante complejas ya que utilizan conocimiento específico. Aún

[2] http://java.sun.com/j2ee/overview.html

así, suele repetirse gran parte de los algoritmos utilizados de unos sistemas a otros. Por lo tanto, se presenta un problema de reutilización de componentes que utilizan conocimiento sobre el domino. Este conocimiento se representa principalmente de forma declarativa en vez de procedimental y será manipulado por distintos sistemas de razonamiento para obtener las soluciones de los problemas. Los PSMs definen un mecanismo para resolver los problemas de los KBS separando el conocimiento del dominio del conocimiento de control que lo utiliza. De esta forma, los PSMs son completamente independientes de un dominio concreto y pueden ser reutilizados para resolver otros problemas cambiando el conocimiento del dominio que manejan. Por ejemplo, un PSM para aplicaciones CBR podría ser el método de recuperación mediante el vecino más próximo. Este método es independiente del dominio y recibe el conocimiento sobre el mismo en forma de medidas de similitud que comparan los casos concretos de la aplicación.

El conocimiento de control de un PSM se determina en términos de las tareas a ser resueltas y las acciones necesarias para resolverlas, modelándose como un proceso abstracto y desligado de un dominio específico.

En Benjamins y Fensel (1998) se describen las características principales de los PSMs:

- Un PSM especifica qué acciones se realizarán para resolver una tarea.
- Un PSM define una o varias estructuras de control sobre esas acciones.
- Los PSMs utilizan el conocimiento del dominio para resolver las tareas.

La ventaja principal de los PSMs consiste en el ahorro de tiempo de desarrollo al utilizar bibliotecas de métodos existentes. Numerosos grupos de investigación vienen utilizando esta metodología desde hace más de dos décadas (McDermott, 1988), sin embargo la definición y creación de PSMs ha resultado ser más complicada de lo que se imaginó en un principio.

Los PSMs juegan un papel fundamental en la implementación de los sistemas de Servicios Web Semánticos actuales, siendo un pilar básico en la organización y representación de la funcionalidad aportada. Aunque conceptualmente sean cosas distintas, ya que los SWS no tienen por qué separar la parte procedimental del conocimiento del dominio, sí que coinciden en varios aspectos involucrados en la representación de la funcionalidad ofertada, su composición y ejecución. El ejemplo más claro se encuentra en la plataforma WSMO que detallaremos en la Sección 2.3.2.2 donde se adaptan directamente las ideas de los PSMs.

De manera genérica, existe un gran consenso en las características básicas de los PSMs (descritas gráficamente en la figura 2.6):

Competencia. La competencia de un PSM es una descripción declarativa

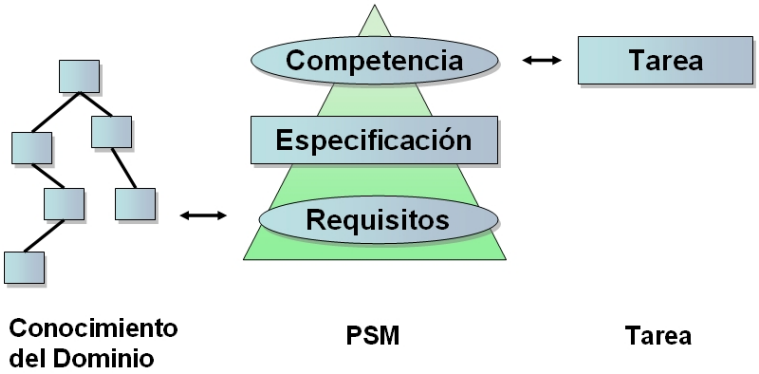


Figura 2.6: Componentes de un PSM y su contexto de aplicación

de su comportamiento. Normalmente, la competencia de un PSM se especifica cómo una *tarea* que debe resolver.

Requisitos. Describen el conocimiento del dominio que necesita un PSM para realizar las tareas indicadas por su competencia.

Especificaciones operacionales. Describe el proceso de razonamiento del método mediante sus entradas/salidas y los distintos pasos ejecutados.

Coincidiendo en estas características se desarrollaron distintos sistemas que utilizaban PSMs aunque sin un lenguaje de especificación común que permitiera su intercambio. Para solucionar este problema se definió el Lenguaje Unificado de Métodos de Resolución de Problemas (Fensel et al., 2003), conocido como UPML (del inglés *Unified Problem-Solving Method Language*). Este lenguaje se ha convertido en un estándar utilizado por muchos grupos de investigación y es utilizado en varias plataformas de SWS.

UPML es un lenguaje especializado en la descripción de PSMs cuyos objetivos son: desacoplar los elementos del modelo, encapsular dichos elementos y modelar explícitamente sus interacción. Para alcanzar estos objetivos se definieron los siguientes elementos:

Tareas. Las tareas describen los objetivos que se deben cumplir para solucionar un problema. UPML añade a esta descripción una segunda parte donde se definen las suposiciones (*assumptions*) sobre el conocimiento del dominio y las precondiciones de la entrada. Estas descripciones se realizan de forma independiente del dominio de la aplicación.

En sentido abstracto, las suposiciones y precondiciones pueden ser vistas como las entradas de la tarea. Sin embargo, se realiza una distinción

entre los datos a procesar (la entrada) y el conocimiento que caracteriza a los KBS. Las precondiciones son condiciones sobre entradas dinámicas, mientras que las suposiciones son condiciones sobre el conocimiento utilizado por el PSM para resolver la tarea. Estas últimas pueden ser comprobadas durante la creación del sistema, sin embargo, las precondiciones se comprobarán en tiempo de ejecución.

Desde el punto de vista sintáctico, una tarea UPML se describe mediante la ontología utilizada, la entrada y salida, y descripciones lógicas del objetivo, precondiciones y suposiciones.

PSMs. Son los encargados de definir los procesos de razonamiento. UPML diferencia entre dos tipos de PSMs: complejos, que descomponen una tarea en varias subtareas; y primitivos, que realizan suposiciones sobre el conocimiento del dominio para realizar un paso del razonamiento.

Los PSMs se describen mediante varios elementos donde destacan la ontología utilizada, la especificación operacional que define el flujo de control y la descripción de las competencias. Respecto al flujo de control destaca su representación mediante el lenguaje MCL (Fensel et al., 1998). La descripción de las competencias ofrece una especificación funcional del PSM que incluye precondiciones, postcondiciones, entradas y salidas.

Modelo del dominio. El modelo del dominio se necesita para definir completamente las tareas en una determinada aplicación y permitir que los PSMs realicen su procesamiento. Este conocimiento se utiliza para especializar las tareas y métodos en un dominio o aplicación concreta, ya que estos dos componentes se especifican de manera genérica.

Ontologías. Ofrecen la terminología independiente del dominio utilizada en las tareas, PSMs y definiciones del dominio. UPML no obliga a utilizar ningún lenguaje específico para representar las ontologías y los axiomas (pre-/postcondiciones). Sin embargo, se ofrecen dos tipos de lenguajes: un lenguaje lógico basado en $(ML)^2$ y MCL (Fensel et al., 1998) y otro lenguaje basado en marcos derivado de CML y KARL (Kifer et al., 1995).

Adaptadores. Se utilizan para ajustar las partes reutilizables al sistema particular que se está implementando. Mediante estos componentes se relacionan las diferentes partes de la arquitectura. Por ejemplo, permite ajustar un PSM para que resuelva una tarea o cambiar las entradas y salidas de los componentes para que encajen entre ellos.

UPML ha sido adoptado por distintos proyectos involucrados en la investigación sobre Servicios Web Semánticos. Aunque los detalles sobre estas

arquitecturas se incluyen en la siguiente sección, expondremos aquí su relación con UPML ya que en muchos casos los SWS son una evolución de esta arquitectura.

ODESWS Gómez-Pérez et al. (2004) es una plataforma de SWS cuya arquitectura se basa en los componentes descritos por UPML (tareas, PSMs, modelo del dominio, ontologías y adaptadores). A la hora de convertir los PSMs en servicios Web no se ha utilizado la sintaxis ni los lenguajes propuestos por UPML, sino que se ha utilizado OWL para representar las ontologías y OWL-S (descrito en la Sección 2.3.2.1) para describir los servicios.

Otro proyecto que ha mezclado UPML con SWS es CHIME (Aroyo et al., 2004). Este proyecto utiliza los servicios Web para compartir contenidos hipermedia. La arquitectura interna de los componentes vuelve a basarse en UPML, aunque para representar las ontologías se delega en XML, RDF y OWL.

Como se puede comprobar en los dos proyectos anteriores, la principal desventaja de UPML se encuentra en los lenguajes de descripción elegidos. En su especificación (Fensel et al., 2003) se comenta que sería mucho más ventajoso utilizar un estándar como OWL para representar las ontologías, por lo que se deja abierta esta parte. Sin embargo a la hora de definir el comportamiento de los PSMs se fija el lenguaje a utilizar (MCL, KARL,...), lo que puede resultar poco conveniente en algunas arquitecturas que quieran utilizar otro tipo de lenguajes lógicos.

Los problemas anteriores han conducido a diversos sistemas de servicios Web basados en UPML a utilizar sus propios lenguajes de representación o, en los mejores casos, a mezclar partes de UPML con sus formatos. Sin embargo, la mayoría siguen el esquema de arquitectura propuesto por UPML.

Este es el caso de jCOLIBRI donde resulta mucho más conveniente utilizar el lenguaje OWL para representar las ontologías del dominio y los propios componentes del armazón. Esta conveniencia radica en la implantación de OWL como estándar de facto y la existencia de diversos razonadores que permiten realizar inferencias sobre las ontologías representadas mediante este lenguaje.

Desde nuestro punto de vista, lo más importante es la representación de las tareas, métodos, pre-/postcondiciones, entradas y salidas. Como veremos a continuación las distintas arquitecturas de SWS (WSMO o OWL-S) mantienen en esencia estos conceptos y son más adecuadas para la plataforma jCOLIBRI al utilizar estándares mucho más extendidos y aplicables como puede ser OWL.

2.3.2 Los Servicios Web Semánticos

El campo de los Servicios Web Semánticos ha desarrollado numerosos estándares de representación de métodos y componentes reaprovechando los estándares y capacidades desarrollados en el área de la Web Semántica. A grandes rasgos podríamos decir que esta línea de investigación se ha encargado de estudiar en profundidad los requerimientos semánticos a la hora de expresar la funcionalidad de los métodos (denominados servicios) y ha utilizado los lenguajes ontológicos de la Web Semántica para describirlos. La mayoría de les requisitos semánticos que aparecen en los estándares de Servicios Web Semánticos se heredan de los PSMs y más concretamente del lenguaje UPML. Como se irá describiendo a lo largo de esta sección, los distintos estándares en este área comparten muchos de los elementos introducidos al describir UPML.

Si los SWS heredan muchas características de los PSMs podemos establecer un claro paralelismo con la evolución de jCOLIBRI y los motivos por los que se aplican estas tecnologías en el armazón, ya que en jCOLIBRI 1 se utiliza una descripción de los métodos basada en los PSMs y UPML, mientras que en la segunda versión se aplican las ideas y metodologías de los SWS. Desde nuestro interés por la composición de sistemas CBR hemos utilizado los avances en este área para describir los métodos y aplicaciones que pueden ser desarrollados en jCOLIBRI 2, tal y como se presentará en el Capítulo 6.

El origen de los SWS se basa en que las ontologías también pueden utilizarse para describir contenido dinámico (como métodos o servicios) aunque normalmente se utilicen para describir contenido estático (datos). De esta forma, los agentes software podrán ser capaces de entender la funcionalidad ofrecida por un servicio e invocarlo automáticamente. Como las ontologías juegan un papel muy importante en el diseño de nuestro armazón la elección de estos estándares parece muy apropiada para definir su arquitectura.

Sin embargo, las arquitecturas de Servicios Web Semánticos han sido diseñadas para funcionar en entornos distribuidos como Internet, por lo que en este trabajo vamos a dejar de lado este aspecto y nos centraremos en las representaciones de los servicios. Desde el punto de vista de la composición de métodos CBR, la posibilidad de ejecución remota es secundaria y lo más importante es estudiar cómo se representa la funcionalidad de cada servicio. En el mecanismo de composición de métodos de jCOLIBRI utilizaremos una representación basada en los estándares de los Servicios Web Semánticos aunque sin tener en cuenta su naturaleza distribuida. De esta forma reaprovechamos el esfuerzo realizado en este campo sobre la representación de métodos y dejamos abierta la posibilidad de ejecución distribuida al estar utilizando aproximaciones compatibles.

Actualmente existen dos plataformas distintas para el desarrollo de SWS: OWL-S y WSMO. A continuación describiremos el tipo de representación uti-

lizada en cada aproximación. A partir de ellas, en el Capítulo 6 se presentará una arquitectura de composición de métodos CBR que toma ideas de ambas plataformas.

2.3.2.1 OWL-S

Esta es la propuesta del consorcio W3C donde se propone utilizar las posibilidades del lenguaje OWL. A partir de esta idea crearon el estándar OWL-S (The OWL Services Coalition, 2004), donde se especifica en forma de ontología la información necesaria para utilizar los Servicios Web Semánticos.

En el diseño de OWL-S se han tenido en cuenta principalmente tres aspectos:

Localización automática de los servicios. Consiste en permitir que los clientes puedan encontrar el servicio adecuado a sus necesidades. Para ello se establece una descripción del servicio interpretable por los clientes software.

Invocación automática de los servicios. Los clientes deben ser capaces de ejecutar automáticamente los Servicios Web. Para ello OWL-S define una interfaz declarativa, interpretable por las aplicaciones software, que incluye una descripción semántica de los distintos argumentos del servicio así como los valores devueltos por éste. Los clientes deberían ser capaces de interpretar esta información para entender qué entradas son necesarias para invocar el servicio y qué información les va a ser devuelta. OWL-S, junto con las ontologías dependientes del dominio, provee los mecanismos necesarios para realizar este tipo de invocación automática.

Composición automática de servicios Web. Esta tarea comprende la selección, composición y interoperabilidad entre diferentes servicios ya que en determinadas situaciones las tareas a resolver pueden ser demasiado complejas como para utilizar un sólo servicio Web. Los clientes deberían ser capaces de manipular estas representaciones junto con la especificación de la tarea a realizar para alcanzar su objetivo automáticamente. Para ello OWL-S define especificaciones de los prerrequisitos y consecuencias de los servicios junto con un lenguaje de descripción de su composición y control del flujo de las invocaciones.

A continuación describiremos la ontología de OWL-S. A alto nivel, esta ontología se diseñó para responder tres preguntas básicas en los servicios Web.

- ¿Qué ofrece el servicio a los clientes? La solución viene dada por el *Perfil* (Profile) utilizado para publicar el servicio.

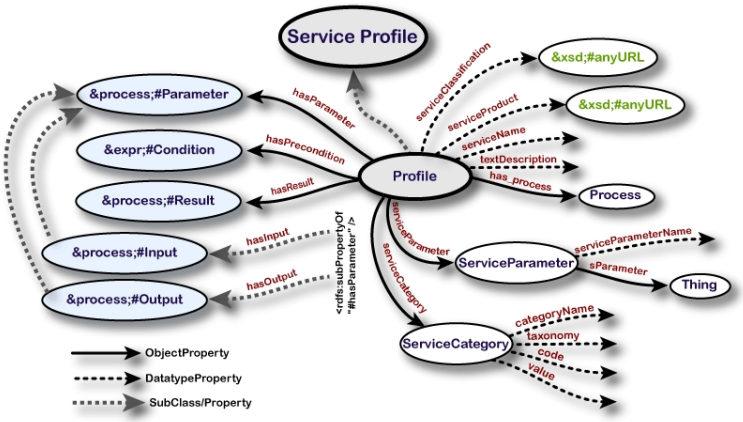


Figura 2.7: Descripción de los Perfiles en OWL-S

- ¿Cómo se utiliza el servicio? La respuesta se especifica el el *Modelo de Proceso* (Process Model).

- ¿Cómo se interactúa con el servicio? Su respuesta aparece en el *Enlace al Servicio* (Grounding). Esta parte define los detalles acerca de los protocolos de transporte de los datos.

El Perfil se encarga de describir los servicios. Realmente no se obliga a utilizar ningún tipo de representación. Sin embargo, las posibilidades de OWL permiten la creación de representaciones especializadas de los servicios. Un perfil de OWL-S describe un servicio en función de tres tipos básicos de información: qué organización ofrece el servicio, de qué función se encarga y qué características lo definen. La parte más importante es la encargada de especificar la funcionalidad del servicio. Aquí aparecen las entradas, salidas, precondiciones y efectos (llamadas IOPEs del inglés Inputs, Outputs, Preconditions and Effects). La Figura 2.7 representa los campos del Perfil.

La ontología del Perfil del servicio no establece ningún esquema para definir las IOPEs, aunque dicho esquema existe en la ontología del *Modelo de Proceso* discutida posteriormente. Las propiedades más importantes del Perfil son las involucradas en la definición de IOPEs: hasParameter, hasInput, hasOutput, hasPrecondition y hasResult (mostradas en la Figura 2.7).

Para entender las IOPEs utilizaremos un ejemplo sobre la compra de libros. Un servicio OWL-S de este tipo tendría como entrada el ISBN del libro a comprar y las credenciales (login y password) del usuario que realiza la compra. La salida podría ser simplemente un justificante de compra con el precio del libro. Para que este servicio pueda ser invocado automáticamente debería cumplirse la precondición de que el usuario tuviera una cuenta válida

en el sistema y el efecto de invocar el servicio sería que el libro sería quitado del stock y enviado al usuario. Ahora supongamos que el usuario utiliza una moneda distinta que el sistema (por ejemplo euros en lugar de dólares). En este caso el agente que invoque el servicio de compra de libros debería componerlo con otro que cambiara el precio del justificante devuelto a la moneda del usuario. Para ello, el tipo de datos de la salida del primer servicio deberá ser compatible con la entrada del segundo. Esta compatibilidad se comprobará mediante una ontología con información sobre los tipos de datos manejados por los servicios web.

Una vez que el servicio ha sido seleccionado, el Perfil deja de tener utilidad. A partir de este punto el cliente utiliza el Modelo de Proceso para controlar la interacción con el servicio. Aunque el Perfil y el Modelo de Proceso tienen diferentes papeles durante la invocación de servicios Web, ambos albergan dos representaciones distintas del mismo servicio. Por lo tanto ambos contendrán una misma especificación de las IOPEs. Como se comentó anteriormente, el Perfil no obliga a ninguna representación concreta de ellas, aunque el Modelo de Proceso sí que lo haga a través de su subclase "Process". En la práctica, el Perfil utilizará la representación del Modelo de Proceso.

La representación de las precondiciones y efectos es una de las partes más difíciles de implementar en un lenguaje basado en XML. La solución de OWL-S consiste en dejar abierta la posibilidad de utilizar cualquier tipo de lenguaje lógico. Para ello se define una clase "Expresion" compuesta por la definición del lenguaje "expresionLanguage" y el cuerpo de la expresión "expresionBody". De esta forma, OWL-S permite utilizar cualquier lenguaje lógico en sus servicios Web.

En OWL-S se definen tres tipos de procesos web:

Atómicos: Servicios directamente invocables, sin subprocesos y ejecutables en un único paso. Es decir, reciben un mensaje de entrada, se ejecutan y devuelven la salida. Para cada servicio atómico hay que proporcionar un Enlace del Servicio (*Grounding*) donde se especifican los detalles de intercomunicación con el mismo.

Simples: Son servicios no invocables que no llevan asociado ningún *Grounding*. Este tipo de servicio representa una abstracción de otros servicios. Son utilizados para proporcionar una visión más sencilla o especializada de un servicio atómico o varios servicios compuestos.

Compuestos: Los servicios compuestos están formados por otros servicios que a su vez pueden ser compuestos o no. Su descomposición puede ser especificada utilizando estructuras de control como secuencias, condicionales, bucles, etc. Las diferentes estructuras de control que proporciona OWL son:

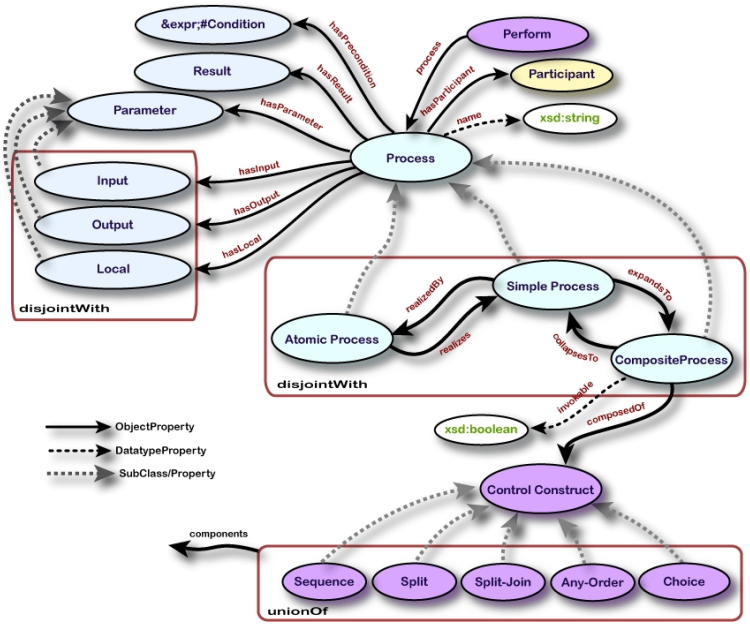


Figura 2.8: Descripción de los Modelos de Proceso en OWL-S

- Sequence. Lista secuencial de servicios.
- Split. Conjunto de servicios que se ejecutan concurrentemente.
- Split+Join. Conjunto de servicios que se ejecutan concurrentemente pero con sincronización. Es decir, esta estructura de control acaba cuando todos sus procesos han terminado.
- Any-Order. Conjunto de servicios ejecutables secuencialmente en cualquier orden (sin concurrencia).
- Choice. Ejecuta uno de los servicios.
- If-Then-Else. Estructura similar a la de los lenguajes de programación. Si se cumple una condición se ejecuta un proceso y sino se ejecuta otro.
- Iterate. Ejecuta los procesos iterativamente hasta que se cumple una condición. Esta estructura es abstracta y es extendida por Repeate-While y Repeat-Until (repetir mientras y repetir hasta).

Obviamente, OWL-S incluye diversos mecanismos para propagar las salidas de un servicio al siguiente. En la figura 2.8 se muestra un diagrama con todos los aspectos del Modelo de Proceso.

La última parte de la ontología de OWL-S es el Enlace del Servicio (Grounding). Esta parte especifica los detalles de acceso al servicio: protocolos, men-

sajes, formatos, serialización, transporte, direccionamiento. El Grounding es una asociación entre la definición abstracta del servicio descrita en el Perfil y el Modelo y un protocolo concreto para el intercambio de información. Debido a la existencia de un amplio trabajo en este área, el consorcio W3C decidió utilizar el protocolo WSDL (*Web Services Description Language*) como lenguaje para el Grounding.

2.3.2.2 WSMO

El estándar WSMO (Web Services Modelling Ontology - Roman et al. (2005)) ha sido desarrollado por diversos investigadores del área de la Web Semántica e Inteligencia Artificial, formando el WSMO Working Group [3]. WSMO define la arquitectura mediante diversas especificaciones ontológicas para los servicios Web y se integra dentro del framework WSMF (Web Services Modeling Framework) que incluye tanto un lenguaje para la descripción de los servicios Web (WSML: Web Services Modeling Language [4]) como una plataforma de ejecución (WSMX: Web Services Execution Environment [5]). Las características de toda la plataforma no son importantes en esta revisión por lo que sólo se detallarán los aspectos más relevantes de WSMO y WSML respecto a la representación de la funcionalidad de los servicios.

El modelo de WSMO se fundamenta en cuatro componentes principales:

Ontologías. Suministran la semántica formal de la información utilizada por los otros componentes. Son el pilar básico de WSMO ya que definen la terminología (dependiente del dominio) para describir a otros elementos. Las ontologías WSMO se representan utilizando el lenguaje WSML y sirven para dos propósitos: definir la semántica formal de la información y enlazar la terminología máquina con la humana. WSMO especifica las siguientes partes de la descripción de una ontología: conceptos, relaciones, funciones, axiomas e instancias. Los conceptos quedan definidos por su jerarquía de subsumción y sus atributos, las relaciones describen interdependencias entre un conjunto de parámetros donde el dominio de los parámetros puede ser un tipo de datos o un concepto, las funciones son relaciones unarias que deben ir acompañadas de un axioma que defina sus dependencias funcionales, las instancias son concreciones de los conceptos o relaciones, y por último los axiomas representan expresiones lógicas para formalizar el conocimiento sobre un dominio específico.

Servicios Web. Los Servicios Web representan una pieza atómica de funcionalidad que puede ser utilizada para construir otras más complejas.

[3] http://www.wsmo.org
[4] http://www.wsmo.org/wsml
[5] http://www.wsmx.org

En ellos se representan los aspectos funcionales que deben ser semánticamente descritos para alcanzar su utilización semiautomática.

Los servicios Web están descritos en WSMO mediante: propiedades no funcionales, ontologías importadas, mediadores, variables compartidas, precondición, postcondición, suposiciones y efectos. Las cuatro últimas características se definen mediante axiomas en WSML ya que representan expresiones lógicas.

La precondición representa las condiciones de información necesarias para ejecutar el servicio (parámetros de entrada). La postcondición establece el servicio ofrecido al usuario. Las suposiciones son las condiciones sobre el estado del mundo que deben cumplirse para poder ejecutar el servicio. Por último, los efectos representan la forma en que el servicio modifica el estado del mundo. Hay que destacar que las suposiciones y efectos definidos en WSMO no se comprueban a la hora de ejecutar el servicio, por lo que pueden ser vistos como un tipo de restricción débil que se recomienda cumplir antes de su invocación.

Aparte de estas características funcionales, los servicios web incorporan una descripción del patrón de comunicación necesario para utilizarlos denominado coreografía del servicio. Es en esta parte donde se definen las entradas y salidas.

Objetivos. Describen aspectos relacionados con los deseos del usuario respecto a la funcionalidad solicitada, contrariamente a la descripción del servicio Web donde se describen las capacidades del servicio.

Los objetivos describen la tarea que un cliente puede llevar a cabo utilizando un servicio Web. Podría decirse que son la perspectiva del usuario de los servicios Web. De esta forma se desacoplan las peticiones de los usuarios y la funcionalidad ofrecida por el servicio Web. Cuando la aplicación cliente define un objetivo, el localizador de servicios Web detectará los servicios disponibles para alcanzarlo. La parte más importante de este elemento es la capacidad o habilidad del objetivo, formulado en los mismos términos que los servicios Web. De esta forma se habilita el razonamiento sobre qué servicios resuelven qué objetivo.

Mediadores. Utilizados como conectores, suministran facilidades de interoperabilidad entre los elementos. Los mediadores rellenan huecos estructurales, semánticos o conceptuales que suelen aparecer entre los diferentes componentes.

WSMO hereda claramente la separación entre servicios y objetivos de la arquitectura UPML. En de Bruijn et al. (2005) se expone la relación entre WSWO y UPML ya que la primera es una evolución de la segunda. Si comparamos los elementos de UPML con los de WSMO se observa fácilmente esta herencia. En WSMO, las tareas pasan a ser objetivos, los PSMs a Servicios

Web, las ontologías siguen manteniéndose y los adaptadores evolucionan en mediadores.

Al igual que UPML carece de una representación del flujo de control ya que se basa en la descomposición de tareas, WSMO tampoco lo incorpora en su arquitectura. Esto no quiere decir que no sea posible utilizar flujos de control más avanzados ya que esta tarea se delega en uno de los componentes de la arquitectura WSMX.

Por su parte, el lenguaje de descripción de WSMO (denominado WSML) también se presenta dividido en capas con distinta expresividad (de Bruijn et al., 2006):

- WSML-Core se basa en la intersección de las Lógicas Descriptivas y la Lógica de Horn. Esta es la variante menos expresiva de WSML
- WSML-DL permite la misma expresividad que las Lógicas Descriptivas.
- WSML-Flight es una extensión de WSML-Core donde se incluye un lenguaje de reglas.
- WSML-Rule extiende WSML-Flight con más características de la Programación Lógica.
- WSML-Full representa completamente la lógica de primer orden y engloba las dos variantes WSML-DL y WSML-Rule.

WSML también puede representarse en XML/RDF e incluso OWL. La arquitectura WSMX incluye su propio razonador para utilizar este lenguaje.

2.3.3 La Plataforma J2EE

La arquitectura de jCOLIBRI se basa en ideas tomadas de las arquitecturas de PSMs, OWL-S o WSMO. Sin embargo, hay que recordar que jCOLIBRI no es sólo una arquitectura sino que es también una plataforma software ya que incluye una implementación de dicha arquitectura. Las tecnologías nombradas anteriormente influyen en el diseño de la arquitectura del armazón pero no realizan ninguna aportación a la implementación de la plataforma. Para realizar esta tarea de implementación, la segunda versión de jCOLIBRI se fundamenta en la plataforma J2EE[6] (Java 2 Enterprise Edition) que describiremos en este apartado.

En este punto conviene aclarar las distintas aportaciones desde el punto de vista de arquitectura y plataforma. Empezando por jCOLIBRI 1, esta versión del armazón basaba su arquitectura en los PSMs y no utilizaba ninguna

[6] http://java.sun.com/javaee/

plataforma de referencia. Esto se debía a que los PSMs tampoco tenían una implementación clara de referencia que les sirviera de plataforma. De esta forma podríamos clasificar a los PSMs como una arquitectura sin plataforma. En este grupo también clasificaríamos a OWL-S, mientras que WSMO sí que es una arquitectura que además cuenta con plataforma (denominada WSMX). A la hora de diseñar la arquitectura de jCOLIBRI 2 se tomaron las ideas de las arquitecturas OWL-S y WSMO, mientras que para implementar la plataforma se utiliza J2EE.

J2EE es una de las plataformas más utilizadas para el desarrollo de grandes aplicaciones empresariales y ofrece varias tecnologías y componentes muy útiles que aumentan la eficiencia y compatibilidad de nuestro armazón.

Esta plataforma sigue un modelo de arquitectura basada en el patrón de diseño Modelo-Vista-Controlador (Gamma et al., 1995). Este patrón divide las aplicaciones en tres componentes: el modelo del dominio que contiene la parte lógica de la aplicación, la vista que presenta los datos al usuario y el controlador que realiza la coordinación entre ellos. A partir de esa arquitectura se implementan una serie de tecnologías que permiten crear grandes aplicaciones empresariales siguiendo este diseño. El componente más importante en este tipo de aplicaciones es la información de la aplicación, y normalmente existen varias fuentes de datos de diversa complejidad que deben ser accedidas de una forma estandarizada. Para ello, se genera la tecnología JDO (Java Data Objects) que define un mecanismo de mapeo entre las bases de datos y los objetos de la lógica de la aplicación. Además, los objetos de la lógica deben ofrecer la máxima flexibilidad para dicho mapeo, para su manejo por cualquier método de la lógica, y para su presentación al usuario tanto mediante interfaces web o en aplicaciones independientes. Dicha flexibilidad se consigue a través de los EJBs (Enterprise Java Beans). Los Java Beans son básicamente objetos normales que incluyen métodos con una signatura estandarizada para acceder a sus atributos. De esta forma, las aplicaciones pueden manejar la información contenida en los Java Beans de una manera muy desacoplada. Además, su mapeo con la capa de información (bases de datos) se puede estandarizar y delegar en un componente independiente de la lógica.

A grandes rasgos, la plataforma J2EE se puede describir siguiendo la estructura mostrada en la Figura 2.9 (Alur et al., 2001). La primera capa se encarga de manejar las bases de datos que contienen la información utilizada por la aplicación y mapearla en EJBs mediante la tecnología JDO. En la siguiente capa se implementa la lógica de la aplicación. Es aquí donde los distintos métodos modifican y transforman los datos representados como EJBs. Los EJBs son una extensión de los Java Beans que ofrecen funcionalidad extra para realizar algunas tareas comunes como manejar las sesiones del usuario. Una vez los datos han sido procesados se pasan a la capa de presentación encargada de mostrarlos al usuario. En las aplicaciones empresariales

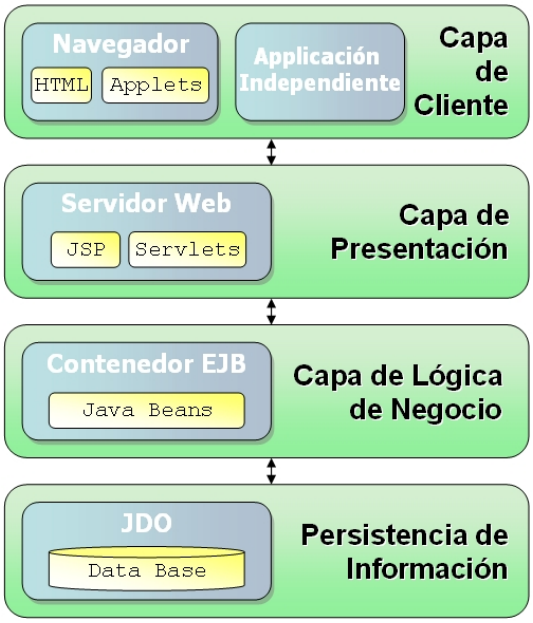


Figura 2.9: Plataforma J2EE

suelen utilizarse servidores web para realizar esta tarea. En ellos se utilizan las tecnologías JSP (Java Server Pages) o Servlets para visualizar los datos y estas tecnologías utilizan internamente los Java Beans para manejar la información. Finalmente, la capa del usuario representa la forma de manejar el sistema tanto a través de un navegador o en una aplicación independiente.

La plataforma J2EE se ajusta mucho a las necesidades de jCOLIBRI y muchos de sus componentes se utilizan en la segunda versión del armazón. Al igual que en las aplicaciones empresariales hay una gran heterogeneidad en las fuentes de datos y se necesita un mecanismo como JDO para estandarizar su acceso, las aplicaciones CBR también deben ajustarse a las distintas formas de almacenamiento de los casos. Dichos casos suelen estar almacenados en bases de datos de distinta naturaleza, por lo que la utilización de la tecnología JDO resulta muy interesante.

Además, en jCOLIBRI2 los datos se representarán como Java Beans para facilitar al máximo su manejo por los métodos CBR del armazón y permitir incorporar las aplicaciones generadas dentro de servidores web.

En conclusión, mediante la plataforma J2EE y sus tecnologías se simplifica el código del armazón al reutilizar componentes que realizan gran parte del trabajo, se garantiza una compatibilidad total con cualquier tipo de representación de los casos en bases de datos y se ofrece la posibilidad de crear

interfaces web de forma sencilla y directa.

2.4 Diseño de Armazones

En las secciones anteriores de este capítulo se ha descrito el dominio de nuestro armazón, las arquitecturas que influyen en su diseño de alto nivel y las plataformas que sirven como base para su implementación. Sin embargo, todavía no se han descrito las características que debe cumplir esta implementación para poder ser considerada como un armazón –mejor dicho, un buen armazón. Este aspecto es muy importante en jCOLIBRI ya que influye directamente en su reusabilidad y facilidad de uso. Estos dos factores son los que decidirán el éxito del armazón dentro de la comunidad CBR y por ello se tomaron muy en cuenta durante todo el proceso de desarrollo.

Para poder desarrollar un buen armazón, primero debemos saber qué son exactamente, qué tipos existen, qué características presentan y qué factores hay que tener en cuenta durante su diseño e implementación. Todas estas cuestiones son las que se detallan en esta sección y se tendrán en cuenta a la hora de presentar las características de jCOLIBRI en el Capítulo 3.

Empezando por la definición, un armazón orientado a objetos, o simplemente armazón, es un diseño reutilizable de parte o de todo un sistema que es representado por un conjunto de clases abstractas y por la forma en que las instancias de sus subclases interactúan (Johnson, 1997). Los armazones son una herramienta de reutilización muy potente para el desarrollo de nuevos sistemas ya que permiten la reutilización tanto de código como de diseño. En consecuencia, el uso de armazones en el desarrollo de nuevas aplicaciones reduce significativamente los costes de producción ya que "usar un armazón es más parecido a mantener una aplicación ya existente que a crear una nueva desde cero" (Shull et al., 2000).

Al igual que con cualquier otro tipo de aplicación, el desarrollo de sistemas CBR es una tarea compleja donde varios de sus componentes pueden ser reutilizados de diseños previos. Añadiendo la creciente necesidad de la comunidad CBR para el desarrollo rápido de prototipos que permitan comparar sus avances con otros investigadores, la necesidad de un armazón CBR parece obvia y ese es el objetivo de este trabajo de tesis.

2.4.1 Características de los Armazones

Independientemente del dominio de la aplicación, existe un conjunto de características que permiten estimar la calidad de un armazón. En teoría, un armazón ideal debería reunir todas estas características. Pero en la práctica es muy difícil desarrollar armazones que cumplan todas ellas debido a que unas características interfieren con otras. Al diseñar un armazón es nece-

sario que se llegue a un compromiso entre las distintas características para alcanzar un diseño lo más flexible y estable posible que además facilite su comprensión por parte de los usuarios. Estas características que deben ser tenidas en cuenta son:

- Facilidad de comprensión y uso. Esta es la característica clave que mide la popularidad del armazón. Un armazón está pensado para ser reutilizado, por lo que si el coste de su uso es elevado no tendrá ningún futuro. Para conseguir esta característica se pide que los mecanismos de extensión del armazón sean sencillos y que las interacciones entre sus clases sean simples y claras. Esta característica también se ve favorecida por una buena documentación que acompañe al armazón.

- Flexibilidad. Los puntos flexibles de un armazón son cada uno de los puntos predefinidos que necesitan ser configurados o extendidos para crear la aplicación (Schappert et al., 1995). Por tanto, cuanto mayor número de puntos flexibles posea el armazón, más aspectos de la aplicación serán configurables. Por el contrario, añadirán mayor abstracción y complejidad al mismo. Por este motivo, es necesario balancear la flexibilidad y facilidad de comprensión del armazón en su diseño.

- Completitiud. Todo armazón es incompleto por definición. Pero cuando en este caso se habla de completitud, se quiere expresar que el armazón ha de poder incorporar el mayor número de implementaciones predefinidas de sus puntos flexibles. La completitud facilita el uso del armazón ya que el usuario podrá elegir entre distintas implementaciones de las funcionalidades más comunes en lugar de tener que reimplementarlas.

- Extensibilidad. Debe ser posible añadir o modificar fácilmente tanto la funcionalidad como las clases del armazón.

Los armazones pueden ser una herramienta realmente útil para la construcción de aplicaciones software. Pero es necesario conocer que no todo son ventajas, que la utilización de armazones también conlleva una serie de inconvenientes que habrá que sopesar e intentar minimizar. Por ello, a continuación describiremos detalladamente las ventajas e inconvenientes de la utilización de armazones.

Entre las ventajas fundamentales podemos citar:

Reutilización de experiencias. Una de las mayores ventajas de los armazón es que las aplicaciones construidas a partir de ellos contienen las ideas de diseño de los desarrolladores expertos que generaron el armazón. Por tanto, los armazones transfieren la experiencia de diseño de sus desarrolladores.

Construcción de software de calidad. El diseño de los armazones está muy pensado y optimizado, por lo tanto las aplicaciones desarrolladas a partir de ellos conservarán la calidad de dicho diseño.

Reducción del tiempo de desarrollo. Una vez que se conoce la forma de utilizar el armazón, el tiempo de desarrollo de una aplicación es mucho menor que implementando desde cero.

Reducción de los costes de mantenimiento. Si para construir una familia de productos similares se usa un armazón concreto entonces el coste de mantenimiento se reduce, ya que todos ellos tendrán una parte común y los encargados de mantenimiento sólo tendrán que conocer en profundidad esa única parte en vez de cada una de las aplicaciones por separado.

Respecto a los inconvenientes, apuntaremos los siguientes:

Incompatibilidades. Un armazón no sirve para construir cualquier tipo de aplicación. Antes de usarlo es necesario conocer qué se puede y qué no se puede hacer con dicho armazón, ya que una vez decidida su utilización pueden surgir incompatibilidades con los requisitos de la aplicación a desarrollar. Si las incompatibilidades no son detectadas a tiempo se producirá un aumento de los costes y del tiempo de desarrollo, por lo que el uso del armazón habrá sido contraproducente.

Falta de control sobre el diseño. Dado que la nueva aplicación se construye tomando como base el diseño definido por el armazón, las decisiones de diseño que toma el desarrollador se encuentran supeditadas a lo impuesto en el armazón. En los armazones el flujo de control de la aplicación está fijado y se encarga de llamar a las clases o métodos creados por el desarrollador de la aplicación. Esto se conoce como *inversión de control* y es un diseño contrario al de las librerías de métodos o componentes donde el desarrollador implementa su propio flujo de control que realiza llamadas a las distintas clases de la librería. La inversión de control ha sido definida en varias ocasiones mediante el llamado *principio de Hollywood*: "no nos llame, nosotros le llamaremos" ("don't call us, we'll call you") (Sparks et al., 1996).

Dificultades de depuración. Depurar una aplicación creada con un armazón puede llegar a ser una tarea compleja. El caso extremo de dificultad de depuración aparece cuando un error en las clases desarrolladas para construir la nueva aplicación afecta a la integridad del armazón. Los errores en el código fuente de la nueva aplicación podrían verse reflejados en las clases del armazón aunque dicho código no haya sido modificado.

Curva de aprendizaje. Antes de usar un armazón hay que aprender a utilizarlo. Como ya se ha mencionado, los armazones son aplicaciones abstractas semicompletas y esta abstracción supone un aumento de la complejidad a la hora de aprender a utilizarlo. El uso eficiente de un armazón puede suponer muchas horas de aprendizaje –para algunos armazones complejos se ha estimado necesario entre 6 y 12 meses (Fayad y Schmidt, 1997)– por lo que es posible que su utilización no resulte rentable en algunas situaciones. Sin embargo, este esfuerzo será amortizado si el armazón se utiliza en varios proyectos.

La curva de aprendizaje es uno de los problemas más importantes que debe abordar un armazón. Por ello debe prestarse una especial atención al proceso de aprendizaje y en consiguiente a la cantidad y calidad de documentación disponible. Normalmente, la documentación de un armazón se realiza por medio de tres elementos: manuales de referencia, tutoriales y aplicaciones de ejemplo.

El *manual de referencia* suele contener información detallada sobre cada una de las clases del armazón. Para cada clase suelen describirse sus responsabilidades, el rol que juega cada uno de sus miembros e información sobre cada uno de los métodos de la clase. Esto último incluye la funcionalidad del método, sus parámetros e incluso sus precondiciones y postcondiciones. Algunos manuales de referencia también incluyen descripciones de los puntos flexibles y de cuáles son las clases y los métodos que configuran dichos puntos flexibles. Los manuales de referencia no son demasiado útiles para aprender a usar un armazón. Es necesario mucho tiempo de estudio del manual antes de poder desarrollar una aplicación pues no guía al usuario, sino que es meramente descriptivo.

Sin embargo, los *tutoriales* sí enseñan cómo crear una aplicación a partir del armazón que documentan. Suelen describir resumidamente los pasos a seguir para desarrollar una aplicación sencilla. El problema de los tutoriales es que suelen ser demasiado escuetos, quedándose en la superficie del desarrollo de aplicaciones y no profundizan en las funcionalidades más complejas que pueda incluir un armazón.

Para resolver esta carencia, las *aplicaciones de ejemplo* suelen consistir en el código fuente de las aplicaciones que los desarrolladores del armazón han construido durante la creación del mismo. Este tipo de documentación es muy económica ya que no se crea ex profeso para documentar sino que las aplicaciones de ejemplo son implementadas durante el desarrollo del armazón para probarlo y depurarlo. Además permite ejemplificar de manera sencilla las capacidades y el uso del armazón. Su inconveniente principal es que las aplicaciones de ejemplo por sí solas son demasiado complicadas para los usuarios menos experimentados (G. y Denommee, 1999).

Algunos armazones también incluyen su código fuente como parte de la

documentación. El código fuente suele ser demasiado complicado de tratar, por lo que es de gran utilidad incluir documentación adicional que permita navegar por este código y conocer la funcionalidad de cada una de sus clases. Actualmente, esta documentación puede ser generada automáticamente a partir de los comentarios incluidos en el código fuente usando herramientas como Javadoc[7], en el caso de armazones implementados en Java, o Doxygen[8], para los implementados en C y C++.

En el caso concreto de jCOLIBRI se ha tenido muy en cuenta la importancia de la documentación y se han incluido todos los tipos de documentación descritos anteriormente. Desde la página web de la aplicación[9] es posible acceder a manuales de referencia, diagramas de clases y secuencia, y un completo tutorial. Además, jCOLIBRI incluye más de 30 aplicaciones de ejemplo que muestran cómo utilizar toda la funcionalidad ofrecida por el armazón. Los detalles de esta documentación se describirán en el Capítulo 3.

2.4.2 Tipos de Armazones

Los armazones suelen clasificarse en dos tipos distintos (Roberts y Johnson, 1997). Los *armazones de caja blanca* (white-box frameworks) se basan principalmente en la aplicación de mecanismos de extensión y flexibilización. El desarrollador de aplicaciones ha de tener un gran conocimiento del armazón para poder crear nuevas subclases o extender aquella funcionalidad que pueda ser útil para su aplicación. Este tipo de armazón es el más sencillo de diseñar pero el que necesita de una mayor esfuerzo de comprensión por parte del usuario.

En el otro grupo están los *armazones de caja negra* (black-box frameworks). Aquí el mecanismo fundamental de reutilización es la composición y parametrización. El desarrollador de aplicaciones sólo ha de seleccionar algunos de los componentes predefinidos del armazón, parametrizarlos y conectarlos. Son los más sencillos de comprender y los que menor esfuerzo suponen al usuario. En cambio, son menos flexibles que los anteriores y los más complejos de diseñar. La evolución natural de un armazón comienza con un diseño de caja blanca que acaba en una arquitectura de caja negra. El armazón de caja negra resultante suele tener un constructor o compositor asociado que genera el código de la aplicación. Este compositor visual permite al diseñador de la aplicación realizar la conexión y configuración de los distintos componentes del armazón.

Los armazones no son la única técnica de reutilización de software. Existen otras técnicas que pueden ser utilizadas en su lugar o combinadas. Dichas

[7] http://java.sun.com/j2se/javadoc/
[8] http://www.stack.nl/ dimitri/doxygen/
[9] http://gaia.fdi.ucm.es/projects/jcolibri/

técnicas presentan una serie de características distintas a los armazones:

- Bibliotecas de clases. Son conjuntos de clases capaces de cubrir una funcionalidad concreta. Las bibliotecas de clases permiten fundamentalmente la reutilización de código y son independientes del domino. Por el contrario, un armazón supone además la reutilización de un diseño para la implementación de aplicaciones en un dominio concreto. Las clases de una biblioteca pueden ser tratadas y comprendidas por separado, mientras que un armazón ha de ser tratado como un todo, ya que sus clases han sido implementadas de tal modo que definen la arquitectura de un conjunto de aplicaciones. Ya que el armazón define la arquitectura, es él quien se encarga del flujo de control, produciendo la inversión de control mencionada anteriormente. En cambio, la reutilización de bibliotecas de clases supone que la aplicación desarrollada se encargará de dirigir el flujo de control, haciendo las llamadas oportunas a las clases de la biblioteca.

  Actualmente, el auge de las aplicaciones web hace necesario un equilibrio entre ambas aproximaciones, o dicho de otra manera, que el flujo de control pueda ser definido tanto por el armazón como por la aplicación. El flujo de control en las aplicaciones web no puede estar fijado por la aplicación sino que está definido por las distintas acciones y elecciones del usuario. Por lo tanto surge una incompatibilidad para el uso de armazones en aplicaciones web. Obviamente, los armazones no pueden limitarse simplemente a aplicaciones independientes y dejar de lado las aplicaciones web ya que supondría una limitación enorme. Por lo tanto, es necesario dejar abierta la puerta del control del flujo cuando se diseña un armazón que pretenda ser utilizado para crear aplicaciones independientes y aplicaciones web.

- Generadores de aplicaciones. Los generadores de aplicaciones se basan en la utilización de lenguajes de alto nivel específicos de dominio para construir aplicaciones software. Mediante un compilador, estos lenguajes pueden generar gran parte del diseño de la aplicación. Dicho esqueleto deberá ser completado posteriormente por el diseñador. En los armazones este esfuerzo de completado de código se reduce ya que el nivel de abstracción es menor. La principal diferencia entre ambas aproximaciones reside en que los generadores suelen usan una sintaxis propia que el usuario debe conocer mientras que los armazones utilizan lenguajes de programación de uso extendido.

- Patrones de diseño. Los patrones de diseño permiten la reutilización de información de diseño. Consisten en proponer una solución a problemas de diseño que se repiten frecuentemente en un determinado contexto. Los patrones son más abstractos que los armazones, ya que sólo describen ideas de diseño. Además, su envergadura es distinta ya que los

patrones resuelven problemas de diseño a nivel de micro-arquitecturas mientras que los armazones lo hacen a nivel de arquitectura global de la aplicación. Aunque son conceptos diferentes, armazones y patrones están relacionados. Por un lado, los armazones más populares han sido diseñados haciendo uso de patrones. Por otro, los armazones son ejemplos del uso de los patrones de diseño. De hecho, uno de los catálogos de patrones software más importantes (Gamma et al., 1995) fue creado a partir del estudio de distintos armazones.

En el diseño de jCOLIBRI se han mezclado todas estas aproximaciones. Aunque se mantengan las características de los armazones, durante el diseño de nuestro armazón se ha tenido muy en cuenta la compatibilidad web ya que hoy en día es uno de los elementos más importantes para el éxito de cualquier aplicación. Por lo tanto, se permite que el control del flujo sea definido por el desarrollador y se ofrece una funcionalidad más cercana a la de las bibliotecas de componentes.

Además, jCOLIBRI también incorpora la posibilidad de generar las aplicaciones. En la primera versión del armazón no se utiliza ningún lenguaje específico ya que se sigue un proceso de descomposición de PSMs y el código se genera a partir de dicha descomposición. Sin embargo en la segunda versión se utiliza todo un sistema de plantillas descritas de forma muy similar al lenguaje de control de flujo de OWL-S que permite la generación del código de las aplicaciones CBR. Todas estas características se expondrán en sus correspondientes capítulos.

Por último, destacar la numerosa inclusión de patrones de diseño en el código de jCOLIBRI. El más importante sería el patrón Modelo-Vista-Controlador ya que se sigue el diseño planteado en la plataforma J2EE como se expuso en el apartado 2.3.3.

2.4.3 Armazones CBR

Además de jCOLIBRI, existen otros esfuerzos dentro de la comunidad CBR para el desarrollo de armazones que posibiliten una implementación rápida y robusta de este tipo de aplicaciones.

CBR*Tools (Jaczynski y Trousse, 1998) es un armazón orientado a objetos implementado también en Java. En este armazón se identifican los siguientes ejes de variabilidad: la delegación de los pasos de razonamiento, la separación del almacenamiento e indexación de los casos, el diseño de índices como componentes reutilizables y el diseño de patrones de adaptación. CBR*Tools mantiene la inversión de control y el desarrollador debe utilizar una serie de puntos de extensión para implementar la aplicación. Su documentación incluye un tutorial y una aplicación de ejemplo.

Orenge (Schumacher, 2002) es una suite de productos comerciales para la recuperación y tratamiento de la información empresarial. Al ser una apli-

cación comercial son muy pocos los detalles sobre su diseño. Orenge es una plataforma basada en componentes donde se utiliza un lenguaje de descripción del flujo de control (implementado en uno de sus componentes) al que se añaden y parametrizan los otros componentes que añaden la funcionalidad.

CAT-CBR (Abásolo et al., 2002) es otro armazón que utiliza UPML para especificar los componentes CBR. Este sistema ofrece una librería de componentes CBR, un lenguaje y editor gráfico para especificar nuevos componentes y una sintaxis específica para implementar el código operacional de los componentes. Por último, se incluye un servicio de búsqueda donde se especifican los requisitos de una aplicación CBR junto con los modelos de dominio existentes y se obtienen los componentes que satisfacen dichos requisitos.

Además, en la comunidad de Aprendizaje Máquina podemos encontrar una gran número de iniciativas dirigidas a la construcción de librerías con métodos reutilizables. MLC++ (Kohavi et al., 1996) ofrece una librería de clases C++ para asistir en el desarrollo de nuevos algoritmos. ANN[10] es una librería también en C++ que incluye estructuras de datos y algoritmos para realizar búsquedas exactas o del vecino más cercano en espacios de búsqueda con un elevado número de dimensiones. Finalmente, Weka (Witten y Frank, 2000a) es una librería gratuita desarrollada en Java que contienen muchas implementaciones de algoritmos de clasificación y predicción numérica. Weka ofrece la posibilidad de utilizarlo como un armazón de caja-blanca o de caja-negra ya que permite extenderlo mediante código o utilizar su propio compositor visual para crear aplicaciones mediante parametrización de componentes. En dicha herramienta de composición se pueden elegir distintas configuraciones del flujo de control ya preestablecidas e ir rellenando cada etapa de dicho flujo con un componente de un tipo determinado. Al elegir un componente se ofrece una interfaz gráfica que permite configurarlo. De esta forma se posibilita la construcción de aplicaciones prototípicas de este dominio.

2.5 Conclusiones

Este capítulo presenta los cuatro grandes bloques sobre los que se fundamenta el trabajo de esta tesis. Teniendo en mente el objetivo de desarrollar una plataforma para la composición de sistemas CBR, lo primero que se realiza es una descripción de este área y de sus características. A continuación se describe qué es la Web Semántica y su aportación más importante: las ontologías. El siguiente bloque de este capítulo se centra en la descripción de las arquitecturas y plataformas que se toman como base, y por último se detallan las características que debe cumplir un armazón como jCOLIBRI sobre el que se construye todo el sistema de composición de métodos CBR.

[10] http://www.cs.umd.edu/ mount/ANN/

En la descripción del Razonamiento Basado en Casos se plantean los componentes involucrados en este tipo de aplicaciones, poniendo especial atención en los casos –que contienen la fuente de experiencia sobre la que se razona. También se detallan las distintas etapas en las que se dividen estas aplicaciones siguiendo el modelo de Aamodt y Plaza (1994): recuperación de los casos más similares a la consulta, reutilización de la solución de dichos casos para adaptarlas al nuevo problema, revisión de la solución propuesta, y aprendizaje de dicha experiencia como un nuevo caso del sistema.

Al existir distintas familias de aplicaciones CBR se incluye una descripción de cada una de ellas ya que jCOLIBRI incorporará métodos especializados que se utilizarán dependiendo del tipo de sistema a desarrollar. Entre estas familias destacan el CBR Textual –encargado de razonar con experiencias representadas mediante lenguaje natural–, el CBR con Conocimiento Intensivo (KI-CBR) donde se añade conocimiento complementario al de los casos, y el CBR Conversacional donde se realiza un diálogo con el usuario para refinar sus preferencias y que da origen a los sistemas CBR de recomendación de productos. Estos tres tipos de aplicaciones CBR serán descritas en los Capítulos 4, 5 y 6 donde se detallarán los métodos que ofrece jCOLIBRI para su implementación.

El segundo concepto descrito en este capítulo son las ontologías. Las ontologías ofrecen una representación semántica del conocimiento que se utiliza en este trabajo de diversas maneras.

La primera aportación de las ontologías es su capacidad para representar el conocimiento complementario de los casos necesario en las aplicaciones KI-CBR. En el Capítulo 5 se presentará una aproximación que permite desacoplar el conocimiento sobre el dominio de los sistemas CBR mediante ontologías que encapsulan dicha información. De esta forma es posible suplir la carencia de conocimiento de recuperación y adaptación cuando los casos no son capaces (o suficientes) para realizar esta tarea.

La otra aportación de las ontologías al trabajo de tesis se basa en la posibilidad de utilizarlas para describir el comportamiento de los distintos componentes de la plataforma. Para ello, se exponen las características de los Servicios Web Semánticos, que utilizan este mecanismo para representar el funcionamiento de servicios accesibles a través de la Web.

Los Servicios Web Semánticos nos sirven para definir la arquitectura de la plataforma que hemos desarrollado, ya que podemos utilizar las distintas tecnologías surgidas en este campo para describir los componentes CBR de nuestro armazón.

Las dos arquitecturas presentadas son OWL-S y WSMO. En ellas se utilizan ontologías para describir el funcionamiento de los servicios web: entradas, salidas, pre-/postcondiciones, etc. Nuestra propuesta se basa en la

reutilización de estas descripciones para representar el comportamiento de los métodos de jCOLIBRI y permitir su composición. Este mecanismo se presentará en el Capítulo 6.

Esta tercera sección del capítulo también presenta los Métodos de Resolución de Problemas por dos motivos. El primero es que sirven como arquitectura de referencia a la primera versión de jCOLIBRI, y el segundo es que son el origen de los Servicios Web Semánticos utilizados en la arquitectura de la segunda versión del armazón.

Desde el punto de vista de la plataforma a utilizar en jCOLIBRI se introduce la tecnología J2EE. Esta plataforma incluye diversos componentes que facilitan la implementación de nuestra arquitectura y sirve como base tecnológica de nuestro armazón.

Una vez descritas las características de los sistemas CBR, y de las arquitecturas, plataformas y tecnologías involucradas en la implementación de jCOLIBRI se presentan también las características que debe cumplir nuestro desarrollo para que sea un buen armazón software. Para conseguir una plataforma de composición de sistemas CBR eficiente, primero necesitaremos crear un armazón que soporte dicha plataforma ofreciendo los métodos (o componentes) necesarios. Por lo tanto en la Sección 2.4 se exponen características de los armazones que se tuvieron en cuenta durante el desarrollo de jCOLIBRI.

El siguiente capítulo se encargará de describir la implementación de jCOLIBRI mostrando las características de su arquitectura y plataforma. Primero se introducirá jCOLIBRI 1 para pasar a describir la segunda y definitiva versión del armazón.

Capítulo 3

jCOLIBRI

– Empezaré con la caja. Si no sale bien a la primera, la mejoraremos hasta que quede perfecta.

En este capítulo se describe en detalle la plataforma jCOLIBRI tanto desde el punto de vista de la arquitectura como de su implementación. Durante el trabajo de tesis se generaron dos versiones distintas de la plataforma denominadas jCOLIBRI 1 y jCOLIBRI 2. La primera versión del armazón se fundamenta en una arquitectura basada en Métodos de Resolución de Problemas (PSMs) que podían componerse a través de una completa interfaz gráfica. Sin embargo esta arquitectura inicial acabó presentando una serie de inconvenientes que desembocaron en la segunda versión del armazón.

La Sección 3.1 presenta la arquitectura e implementación de jCOLIBRI 1 donde se ofrecen una serie de herramientas gráficas para la composición de PSMs que son capaces de generar el código final de la aplicación CBR. Como se expondrá en la Sección 3.1.2, las herramientas ofrecidas permiten además configurar el acceso a las bases de casos, el diseño de la estructura de los casos y la configuración de las distintas medidas de similitud, así como otras características de los sistemas CBR. Aunque estas herramientas eran muy útiles para usuarios diseñadores que no deseaban entrar en los detalles del código del armazón, no resultaban apropiadas para los desarrolladores que quisieran ampliar la funcionalidad de jCOLIBRI extendiendo sus clases y métodos. Esta y otras razones son las que conducen al desarrollo de jCOLIBRI 2 según se expone en la Sección 3.1.3.

jCOLIBRI 2 es una plataforma dividida en dos capas: una orientada a desarrolladores y otra orientada a usuarios diseñadores. La Sección 3.2 presenta la capa para desarrolladores, mientras que la descripción de las herramientas

de composición para diseñadores se retrasa al Capítulo 6 debido a su importancia. De esta forma, la descripción de jCOLIBRI 2 incluida en este capítulo se centra en el diseño de un armazón de caja blanca que facilite su extensión por los programadores y que sirva como base sólida a las herramientas de composición que se ofrecerán a los diseñadores.

Una vez descritas ambas plataformas, la Sección 3.2.2 incluye una recopilación de todos los métodos implementados en el armazón (tanto en la versión 1 como en la versión 2). Esta recopilación nos servirá para presentar una visión global de la funcionalidad ofrecida por jCOLIBRI a sus usuarios.

3.1 jCOLIBRI 1

jCOLIBRI 1 es la primera versión de nuestra plataforma para la composición de sistemas CBR. Esta versión permite el desarrollo de aplicaciones CBR mediante una completa interfaz gráfica que posibilita la composición de los distintos métodos incluidos en el armazón. Una vez que el sistema ha sido configurado, es posible generar el código de la aplicación CBR y ejecutarla o integrarla de forma independiente.

La arquitectura de jCOLIBRI 1 está muy ligada a las ideas de diseño de su predecesora COLIBRI[1], creada por Díaz-Agudo (2002). COLIBRI es un armazón para el desarrollo de aplicaciones CBR (más específicamente KI-CBR) implementado mediante el lenguaje LOOM. Este armazón define claramente la arquitectura de métodos/tareas que detallaremos a lo largo de este capítulo. Sin embargo, el lenguaje de implementación utilizado dificultaba su extensión y popularización. Por ello se realiza su actualización al lenguaje orientado a objetos Java y pasa a denominarse jCOLIBRI.

A lo largo de esta sección describiremos las ideas de diseño de la arquitectura de esta versión del armazón así como la funcionalidad ofertada y algunos detalles de implementación. Finalmente concluiremos con una exposición de los problema de esta arquitectura y que condujeron al desarrollo de la segunda versión de jCOLIBRI

3.1.1 Diseño y arquitectura

jCOLIBRI 1 es la evolución tecnología de la plataforma COLIBRI a los lenguajes de orientación a objetos y nuevas tecnologías de componentes. Actualmente, cualquier herramienta o plataforma de desarrollo pública que quiera conseguir cierta aceptación debe ser implementada en este tipo de lenguajes. En el caso de los armazones este requisito se acentúa debido a su capacidad de extensión: encapsulación, herencia, polimorfismo, etc. Por todo ello, jCOLIBRI se presenta como una plataforma de composición de sistemas CBR donde se aplican y aprovechan todas estas nuevas tecnologías. Aunque el

[1] Cases and Ontology Libraries Integration for Building Reasoning Infrastructures

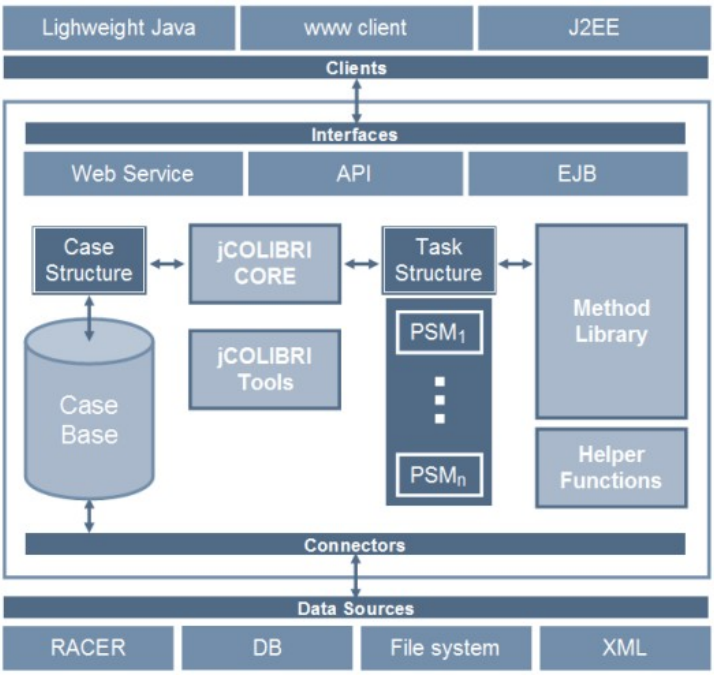


Figura 3.1: Arquitectura global de jCOLIBRI 1

cambio tecnológico es importante, el diseño de esta versión inicial del armazón estuvo muy ligado a las ideas arquitectónicas de su precursor. Por lo tanto, a lo largo de este capítulo se expondrán algunas de sus características.

Para realizar esta exposición se desglosarán los diferentes componentes de la Figura 3.1 donde se muestran los elementos más importantes de la arquitectura de jCOLIBRI 1.

3.1.1.1 Las arquitectura de tareas y métodos

La arquitectura de esta primera versión del armazón se fundamenta en la metodología KADS (Wielinga et al., 1992) y sus descendientes CommonKADS (Van de Velde, 1994) y UPML (Fensel et al., 2003). Es esta última la que más aportaciones realiza y por ello fue detallada en el Apartado 2.3.1. Esta metodología proponía el diseño de los *Sistemas Basados en Conocimiento* mediante modelos abstractos consistentes en componentes reutilizables que contienen Métodos de Resolución de Problemas (PSMs) y ontologías que modelan el dominio.

En COLIBRI dichos elementos se representan mediante una ontología –denominada CBROnto– que describe las diferentes tareas que componen el ciclo CBR. A su vez incluye la terminología sobre CBR necesaria para

describir la funcionalidad de los distintos métodos incluidos en el armazón y así permitir la asignación de cada uno de ellos a la correspondiente tarea que resuelven. CBROnto también sirve para añadir el conocimiento específico sobre el dominio que completa los PSMs con la información necesaria para su instanciación y ejecución.

Al igual que en UPML, la plataforma COLIBRI utiliza tareas simples y compuestas. Las primeras son resueltas mediante un método que implementa una pieza de funcionalidad, mientras que a las segundas se asignan métodos que las descomponen en tareas más simples.

Este diseño basado en tareas y métodos se mantiene en la arquitectura de jCOLIBRI. Sin embargo, se realiza una variación muy importante que consiste en la delegación a un segundo plano de CBROnto. Originalmente esta ontología era un componente fundamental y activo en la plataforma ya que se utilizaba directamente para la generación y ejecución de las aplicaciones CBR. En el cambio al paradigma de orientación a objetos se pierde este papel activo y simplemente se utiliza su jerarquía de conocimiento terminológico sobre CBR para guiar el diseño de la jerarquía de clases. Por lo tanto, parte de su información se reaprovecha ya que sirve para estructurar las clases del armazón, sin embargo se pierden las capacidades de razonamiento que ofrecía la ontología. Gracias a la herencia de CBROnto, la jerarquía de clases se definió inicialmente con un fundamento muy estudiado y eso ha permitido mantener su núcleo estable durante las subsiguientes versiones publicadas. De esta forma se definieron las responsabilidades y signaturas de clases como: *Caso*, *Base de Casos*, *Consulta*, *Similitud*, etcétera. Estas son las clases que forman el componente denominado "jCOLIBRI Core" en la Figura 3.1.

Aunque CBROnto desaparezca como elemento central de la arquitectura sí que se mantiene el diseño basado en tareas y métodos. Las tareas siguen describiendo las distintas etapas del ciclo CBR, y los métodos serán los encargados de resolver dichas tareas. Para ello, se distingue entre dos tipos de métodos: los *métodos de resolución* pueden ser asignados a tareas simples y encapsulan cierta funcionalidad, mientras que los *métodos de descomposición* se encargan de dividir las tareas compuestas en una sucesión de tareas simples. Durante este proceso de asignación de métodos a tareas, los métodos son instanciados por el usuario y parametrizados convenientemente con el conocimiento específico del dominio que requieren.

Las tareas se estructuran jerárquicamente de forma que un conjunto de subtareas descomponga la funcionalidad de la tarea superior. Las principales tareas de esta organización son Retrieve, Reuse, Revise and Retain (Recuperación, Reutilización, Revisión y Retención) siguiendo el famoso modelo presentado por Aamodt y Plaza (1994) y mostrado en la Figura 2.2 (pág. 16). Dichas tareas son descompuestas en secuencias de otras tareas recursivamente hasta que una tarea no puede ser descompuesta en acciones más

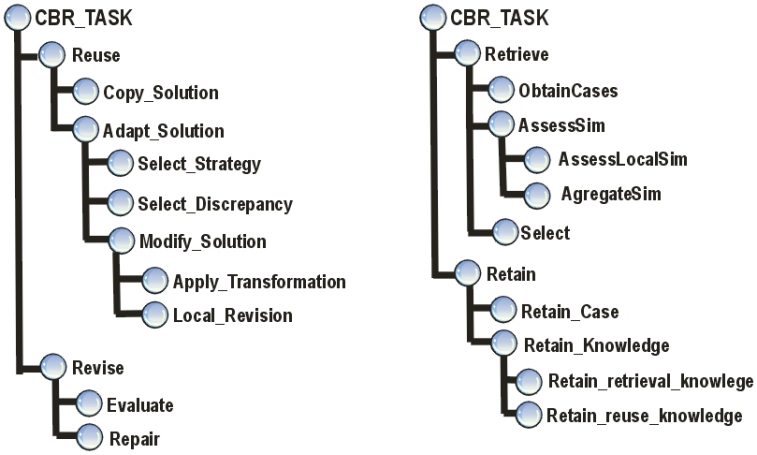


Figura 3.2: Estructura de tareas de CBROnto

sencillas y es resuelta por uno de los PSMs incluidos en el armazón. La Figura 3.2 muestra esta organización en tareas donde se refleja una descomposición prototípica del ciclo CBR. Siguiendo este proceso de descomposición se van asignando PSMs cada una de las tareas y así se obtiene una secuencia de métodos que representa la implementación de un sistema CBR.

El esquema de tareas y métodos de jCOLIBRI 1 sigue el diseño de CBROnto. Sin embargo, la ontología no se utiliza para su representación ya que en esta versión no se aprovechan su capacidad de razonamiento. En su lugar se usa una representación mediante archivos XML que aunque no es tan rica en el aspecto semántico sí que simplifica su tratamiento por parte de la interfaz del armazón.

La información a representar en el caso de las tares consiste simplemente en el nombre de la misma y su descripción. La definición de los métodos es mucho más extensa y en ella se incluye la siguiente información:

- Nombre del método.
- Descripción.
- Precondición. Representa las condiciones necesarias para aplicar el método.
- Tipo de método: descomposición o resolución.
- Parámetros. Sirven para adaptar los métodos al dominio concreto de la aplicación.
- Competencias. Indican las tareas que el método puede resolver.

- Subtareas. Especifican las tareas que un método de descomposición genera.
- Postcondición. Representa el estado de la aplicación tras aplicar el método.

Como puede observarse, esta descripción de los métodos sigue muy de cerca la aproximación de UPML mostrada en la Sección 2.3.1. Dentro de esta representación destacan las pre-/postcondiciones. Aunque la utilización de las pre-/postcondiciones dentro del proceso de composición de métodos nunca llegó a implementarse en esta versión del armazón, ya quedaban reflejadas como un requisito que su representación debía mantener para mejorar el proceso de composición en futuras versiones.

Listado 3.1: Definición en XML de las tareas y métodos en jCOLIBRI 1

```
<Tasks>
    <Task>
        <Name>CBR Task</Name>
        <Description>Main CBR task</Description>
    </Task>
    <Task>
        <Name>Retrieve Task</Name>
        <Description>CBR case retrieval</Description>
    </Task>
    ...
</Tasks>
<Methods>
    <Method>
        <Name>cbrarm.method.CBRMethod</Name>
        <Description>Main CBR method that will divide the CBR process
            in the four typical tasks.
        </Description>
        <ContextInputPrecondition/>
        <Type>Decomposition</Type>
        <Parameters/>
        <Competences>
            <Competence>CBR Task</Competence>
        </Competencies>
        <SubTasks>
            <SubTask>Retrieve Task</SubTask>
            <SubTask>Reuse Task</SubTask>
            <SubTask>Revise Task</SubTask>
            <SubTask>Retain Task</SubTask>
        </SubTasks>
        <ContextOutputPostcondition/>
    </Method>
    ...
</Methods>
```

En el Listado 3.1 se muestra un extracto de la representación real de las tareas y métodos de jCOLIBRI 1. Estos archivos de configuración simplemente definen el comportamiento de los distintos componentes del armazón, delegando en las clases java su implementación. Esta es una de las características más importantes de la arquitectura de jCOLIBRI: la separación entre el proceso de diseño y el proceso de implementación. Esta separación implica dos aproximaciones distintas hacia el usuario dependiendo de cuál sea su perfil: diseñador o desarrollador. De esta forma se amplia el abanico de usuarios potenciales del armazón y se tienen en cuenta sus distintos intereses respecto al proceso de creación de aplicaciones CBR.

Los usuarios diseñadores serán aquellos usuarios que quieran componer sistemas CBR sin detenerse en la programación directa del código. Simplemente están interesados en el diseño conceptual de la aplicación CBR y no están dispuestos a perder tiempo con los detalles de implementación. Sin embargo, los usuarios desarrolladores son aquellos que prefieren o necesitan utilizar directamente el código del armazón para controlar a más bajo nivel el comportamiento de la aplicación.

Mediante la definición de las tareas y métodos en archivos XML se permite que los diseñadores modelen los componentes necesarios para configurar una aplicación CBR aunque no estén implementados. De esta forma, el usuario diseñador puede construir una aplicación CBR mediante la interfaz gráfica de jCOLIBRI 1 definiendo únicamente las tareas que descomponen el ciclo CBR y los métodos necesarios para resolverlas. Una vez realizado el diseño, el desarrollo de los métodos requeridos puede ser delegado fácilmente en un programador que realice su implementación.

Por último, cabe destacar una cuestión técnica importante sobre el intercambio de información de los métodos. En jCOLIBRI 1 el flujo de datos entre los métodos que componen la aplicación CBR se realiza a través del *contexto*. El contexto es una pizarra donde los métodos pueden ir dejando los datos producidos de forma que los siguientes puedan utilizarlos como entrada. El contexto contiene la base de casos y cualquier otro tipo de información que pueda ser generada por cualquier método. Por lo tanto, esta pizarra sirve de eje central para el intercambio de información entre los distintos elementos de la arquitectura.

3.1.1.2 Diseño en capas

La principal característica de la arquitectura mostrada en la Figura 3.1 es la división en capas. De forma similar a la arquitectura J2EE expuesta en el Apartado 2.3.3, jCOLIBRI 1 se divide en una capa de persistencia que representa el almacenamiento físico de los casos, una capa con la parte lógica de la aplicación y una capa final para la interfaz del usuario.

La capa de persistencia engloba las distintas formas de almacenamiento de casos que es capaz de manejar el armazón. Como ya se planteó en el Apartado 2.1 los casos de una aplicación CBR pueden almacenarse en distintos medios heterogéneos que varían desde las bases de datos a representaciones en textos de lenguaje natural o incluso ontologías. Por lo tanto es necesario un mecanismo que permita homogeneizar el acceso a todos estos medios de persistencia. Para ello –y según se describirá en el próximo apartado– se utiliza el concepto de *conector*. Los conectores son clases del armazón que ofrecen una única interfaz para el acceso a la capa de persistencia, sea cual sea el origen de datos a utilizar.

La siguiente capa es la que contiene los elementos lógicos del armazón. Este grupo engloba principalmente a los métodos encargados de manipular los casos e implementar las distintas etapas del ciclo CBR. También incluye las clases responsables de la representación de los casos y su organización en memoria una vez hayan sido cargados desde la capa de persistencia. Por último, esta capa incorpora otros componentes que realizan funciones auxiliares relacionadas con la interfaz gráfica del armazón o con funcionalidad común a varios métodos. Aunque en principio el diseño contempla la posibilidad de utilizar las tecnologías emergentes de servicios web o los EJBs de J2EE, estas características nunca llegaron a implementarse en esta versión del armazón. Sin embargo la segunda versión de jCOLIBRI sí que completa este diseño reutilizando la funcionalidad ofrecida por estas tecnologías.

Finalmente, la arquitectura define una capa de presentación al usuario. Aunque inicialmente los métodos de jCOLIBRI 1 se diseñaron de forma independiente al tipo de interfaz, durante la evolución del mismo se prestó mucha importancia a la interfaz gráfica que permitía su composición. Esto desembocó en un acoplamiento de los métodos hacia interfaces orientadas a aplicaciones independientes (también denominadas standalone o lightweight). Aun así, la parte lógica también permite el desarrollo de aplicaciones con interfaz web aunque con ciertas limitaciones. Por lo tanto, la capa de presentación engloba las distintas interfaces gráficas que podían utilizarse en una aplicación CBR desarrollada con el armazón. En principio esta capa es secundaria ya que cada aplicación tendrá diferentes requisitos de presentación, sin embargo jCOLIBRI 1 ofrece ciertos métodos que facilitan la implementación de la interfaz del sistema CBR.

3.1.1.3 Persistencia de los casos

La capa de persistencia es uno de los elementos más importantes de la arquitectura y ha permanecido invariable a lo largo de todas las versiones de la plataforma al haber demostrado ser un diseño muy acertado. Mediante

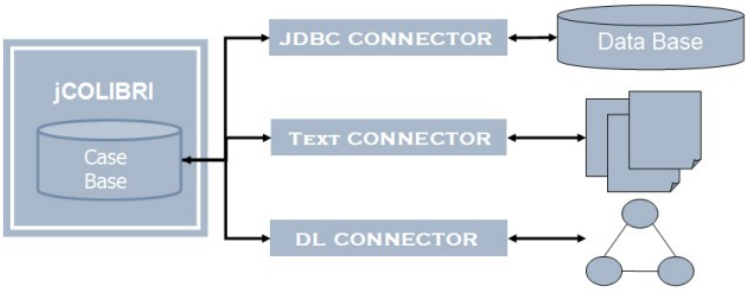


Figura 3.3: El papel de los conectores

los *conectores* se permite un acceso homogéneo a cualquier tipo de medio que almacene los casos. Esto permite aplicar los métodos CBR del armazón en distintos escenarios reales. A grandes rasgos, un conector carga la base de casos a memoria desde cualquier tipo de persistencia y sabe cómo leer y escribir casos en dicho medio. En jCOLIBRI se incluyen varios conectores ya implementados para los medios de persistencia más comunes y además se permite implementar conectores específicos para situaciones donde los casos no siguen un formato de almacenamiento estandarizado.

El papel de los conectores en jCOLIBRI se muestra en la Figura 3.3. Aquí se representan las dos capas conectadas por este componente. Todos los conectores incluidos implementan una misma interfaz para que puedan ser intercambiados de forma transparente en cualquier aplicación. De esta forma se permite un acceso uniforme a bases de datos, ficheros de texto, ontologías,... Si el desarrollador necesitara crear un conector propio simplemente tendría que implementar la interfaz seguida por todos ellos.

Los conectores incluidos en el armazón reciben un fichero de configuración en XML donde se especifican los detalles de la persistencia. Por ejemplo, en el conector de Bases de Datos se indica la URL del sistema gestor de la base de datos, las credenciales de autentificación y el mapeo entre atributos del caso y columnas de la tabla que los almacena.

Al igual que se define una interfaz para los conectores existe otra interfaz para las clases que organizan los casos en memoria y que forman la base de casos con la que trabajan los métodos. Este componente se muestra en la parte izquierda de la Figura 3.3 y es uno de los elementos principales de la capa lógica. De esta forma la arquitectura separa completamente el esquema de persistencia de los casos con el tipo de organización utilizado una vez han sido cargados en memoria.

Tal y como se detalló en la Sección 2.1.3 (pág. 19) existen diversas formas de organizar los casos para optimizar el acceso a los mismos y su proceso de

recuperación. Entre estas aproximaciones se citaban la organización lineal o en árbol. Así pues, nuestro armazón también requería una forma homogénea de acceder a dichas organizaciones tan variadas y por ello se utiliza una interfaz para las distintas clases que implementan la organización en memoria de los casos. De esta forma jCOLIBRI incluye diversas implementaciones de la *base de casos* donde los casos se indexan y estructuran mediante diferentes tipos de datos: listas, árboles, tablas hash, etc. Al igual que en el caso de los conectores, este mecanismo permite que los desarrolladores implementen sus propias organizaciones de la base de casos cuando los requisitos de su aplicación no estén cubiertos por las clases ya incluidas en el armazón.

La combinación de las interfaces de los conectores y la base de casos (`Connector` y `CBRCaseBase`) permiten modificar la persistencia y organización de los casos en memoria de forma cómoda y transparente al resto de métodos. De este modo se ofrece un eje de variabilidad muy importante en el armazón que aumenta el abanico de aplicaciones CBR potencialmente implementables a través de jCOLIBRI.

3.1.1.4 Representación de los casos

Los casos son el elemento central en cualquier aplicación CBR y por lo tanto un armazón como jCOLIBRI debe suministrar un mecanismo de representación flexible y eficiente. Dicho mecanismo debe ser válido para los distintos tipos de representaciones que aparecen comúnmente en este tipo de aplicaciones. Como ya se revisó en la Sección 2.1.2 existen diversas aproximaciones para la representación de los casos que varían desde las listas de pares atributo-valor a otras estructuras mucho más complejas y organizadas.

Para conseguir esta flexibilidad en la representación de los casos, jCOLIBRI 1 utiliza en el patrón de diseño *Composite* (Gamma et al., 1995). Este patrón de diseño organiza los elementos de forma que cada uno de ellos puede estar compuesto por otros, es decir, que a su vez es continente y contenido. Adaptando este esquema a la representación de los casos, los elementos a utilizar se denominan individuos (`Individual`) y representan un atributo del caso. A su vez, cada uno de estos individuos puede estar compuesto de otros individuos. La composición entre individuos se realiza a través de objetos denominados `IndividualRelation` que representan las relaciones entre individuos. Este esquema de representación de los casos se se muestra en la Figura 3.4.

Los atributos que se descomponen en otros atributos se denominan *compuestos*, mientras que los atributos que contienen un valor concreto se denominan atributos *simples*. La principal ventaja de esta aproximación es que permite representar estructuras de casos complejas en forma de árboles de atributos. En estos árboles, los nodos internos son atributos compuestos y las hojas atributos simples.

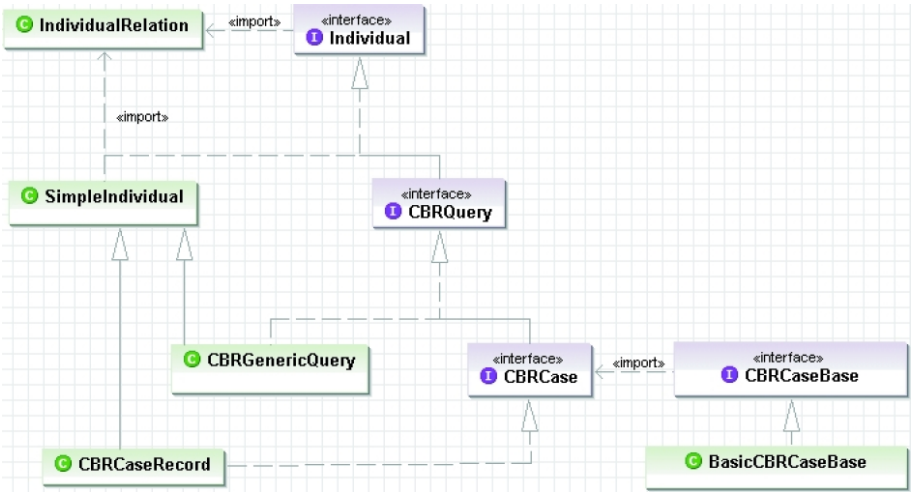


Figura 3.4: Estructura de los casos en jCOLIBRI 1

Cada atributo del caso tiene un tipo concreto: entero, booleano, cadena, etc. jCOLIBRI 1 especifica una interfaz común para todos ellos que facilita su gestión y manejo. Por ejemplo, esta interfaz define el mecanismo de persistencia para los conectores y además permite la generación automática de cuadros de diálogo para obtener sus valores. Esta interfaz también posibilita que los desarrolladores generen sus propios tipos de datos cuando las implementaciones por defecto incluidas en jCOLIBRI no son suficientes.

Dentro de esta organización, las consultas –representadas por la clase `CBRQuery`– son una especialización de la clase `Individual`. Obviamente este individuo estará compuesto por otros individuos que representen los atributos. También es posible especializar la clase `Individual` para definir organizaciones comunes de la estructura del caso. Por ejemplo, la clase `CBRCaseRecord` contiene una implementación por defecto de los individuos compuestos por pares atributo valor. A su vez, los casos –clase `CBRCase`– extienden las consultas para añadir otros individuos que almacenen la solución o el resultado del caso[2].

3.1.2 Funcionalidad y Características

Basándose en la arquitectura descrita en la sección anterior, jCOLIBRI 1 incorpora los métodos más comunes para el desarrollo de aplicaciones CBR. Además se incluye funcionalidad extra agrupada en forma de extensiones. En esta primera versión de la plataforma los métodos se centran principalmente

[2]En esta versión del armazón no se incluye la justificación de la solución como elemento del caso.

en el proceso de recuperación de casos, ya que las etapas de adaptación son muy dependientes del dominio concreto de la aplicación.

Entre la funcionalidad más importante incluida en esta versión del armazón se incluyen dos extensiones que se mantendrán en jCOLIBRI 2. Dichas extensiones incorporan los métodos para desarrollar aplicaciones de CBR Textual (presentados en el Capítulo 4) y los métodos de KI-CBR (presentados en el Capítulo 5). Además, se ofrece una serie de métodos que permiten la evaluación de las aplicaciones generadas con el armazón.

Toda la funcionalidad de jCOLIBRI 1 también se incluye –muchas veces ampliada– en la segunda versión del armazón. Por ello, la descripción de los métodos se delega a la Sección 3.2.2 para realizar una única exposición global de los mismos.

Aunque no se ofrezcan detalles sobre los métodos involucrados, a continuación se describirá el proceso de composición de aplicaciones CBR utilizado en esta versión del armazón y el mecanismo de generación de código a partir de las descomposiciones en tareas y métodos.

3.1.2.1 División del ciclo CBR

En las etapas iniciales del desarrollo del armazón se encontró un gran problema de eficiencia en las aplicaciones CBR generadas. Esta deficiencia se debía a que el ciclo CBR incluía la carga de los casos en memoria y por lo tanto esta tarea se repetía cada vez que se recibía una consulta. Dicha carga de los casos es una de las tareas computacionalmente más complejas del ciclo CBR, requiriendo normalmente bastante tiempo y recursos. Así que, para resolver este problema se decidió dividir las aplicaciones CBR de jCOLIBRI en tres etapas:

- El *preciclo* se encarga de cargar los casos e inicializar los recursos necesarios. Esta parte se ejecuta una única vez antes del ciclo principal.
- El *ciclo* CBR incluye la funcionalidad normal de un sistema CBR. Puede ser ejecutado varias veces con distintas consultas.
- El *postciclo* se encarga de liberar los recursos de la aplicación.

La importancia del preciclo se ha ido corroborando con las sucesivas extensiones del armazón. Como ejemplo más claro podemos citar los métodos de CBR Textual. Este tipo de métodos necesitan procesar el texto para extraer la información y representarla de forma estructurada. Es fácil imaginar que este tipo de proceso puede requerir un largo periodo de cómputo (a veces de varios minutos) por lo que sería inviable ejecutarlo cada vez que se realiza una consulta. Mediante el preciclo se permite adelantar la inicialización y

carga de recursos de forma que se ejecuten los procesos más pesados antes de recibir la consulta.

El ciclo se encarga de implementar la funcionalidad de cada aplicación CBR. Dentro de esta etapa se ejecuta la recuperación, adaptación y almacenamiento de los casos (aunque algunas de estas tareas puedan ser obviadas).

Por último, el postciclo libera los recursos de la aplicación y ejecuta los métodos de mantenimiento. Estos últimos métodos suelen ejecutarse de forma independiente al ciclo CBR, quizás tras la ejecución de cierto número de ciclos, y se encargan de evitar la degradación en eficiencia del sistema según se incorporen nuevos casos que pudieran añadir ruido. Este es el denominado *problema de utilidad* que se presentó en la Sección 2.1.3. Gracias al postciclo, los métodos de mantenimiento quedan incorporados al ciclo CBR de forma natural sin separarlos del proceso de desarrollo de todo el sistema.

Cada una de las tres etapas anteriores se representa como una tarea normal que es resulta por métodos de descomposición en las subtareas apropiadas. Aunque al ejecutar la aplicación estas tres tareas principales se ejecuten independientemente, esta forma de representarlas mantiene la coherencia con el mecanismo de diseño basado en la descomposición de tareas.

3.1.2.2 Interfaz Gráfica

La principal virtud de jCOLIBRI 1 es que ofrece una completa interfaz gráfica para la composición de los métodos del armazón. Esta interfaz se divide en varias herramientas que permiten configurar los distintos componentes involucrados en el diseño de una aplicación CBR:

- Configuración de las tareas y métodos. Esta interfaz permite editar gráficamente los archivos de configuración XML descritos en la Sección 3.1.1 (más concretamente en el Listado 3.1). Estos archivos definen las tareas del sistema y describen los métodos capaces de solucionarlas.

- Composición de tareas y métodos. La Figura 3.5 muestra la ventana de asignación de métodos. En la parte izquierda se puede observar la descomposición en tareas de la aplicación. En las raíces de esta jerarquía aparecen los tres elementos que conforman las aplicaciones en jCOLIBRI: preciclo, ciclo y postciclo. Estas tareas iniciales se descomponen por medio de métodos de descomposición en tareas más simples que pueden ser resultas por los métodos de resolución.

  Al seleccionar una tarea, la parte derecha de la ventana muestra los métodos disponibles en el armazón que pueden asignarse para resolverla. Para obtener esa lista de métodos se utiliza la información almacenada en los archivos XML que contienen la configuración de los métodos y tareas. A partir de la descripción de cada método, el usuario puede

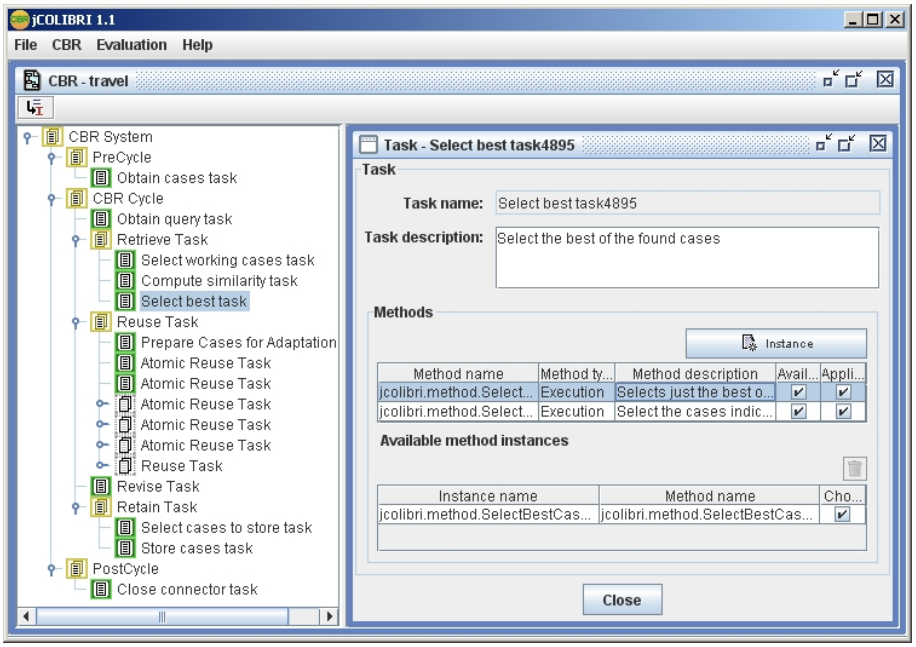


Figura 3.5: División en tareas de jCOLIBRI 1

elegir el más adecuado para la resolución de la tarea e instanciarlo. En el proceso de instanciación también se utiliza la información sobre los parámetros del método almacenada en los ficheros XML y se genera automáticamente un cuadro de diálogo que permite especificar los valores de dichos parámetros. Una vez que se ha añadido esta información, el método se instancia y queda listo para su utilización dentro de la aplicación. Mediante este esquema, la interfaz gráfica permite crear diversas instancias de un mismo método que pueden ser aplicadas indistintamente para resolver la tarea.

- Configuración de los conectores. A través de esta herramienta es posible configurar los distintos conectores que acceden al medio de persistencia que contiene los casos. Esta herramienta permite configurar el acceso a bases de datos, ficheros de textos y ontologías generando el fichero XML que recibe cada uno de ellos. Como ejemplo, la Figura 3.6 muestra el proceso de configuración del conector de bases de datos. En este caso, el usuario debe configurar el método de acceso al SGBD y mapear cada atributo del caso con la columna correspondiente de la tabla.

- Configuración de la estructura del caso. Esta herramienta permite crear gráficamente la estructura de los casos. Esta información se guardará en un fichero XML que es utilizado por los métodos del framework que necesiten conocer dicha estructura. La interfaz muestra los tres com-

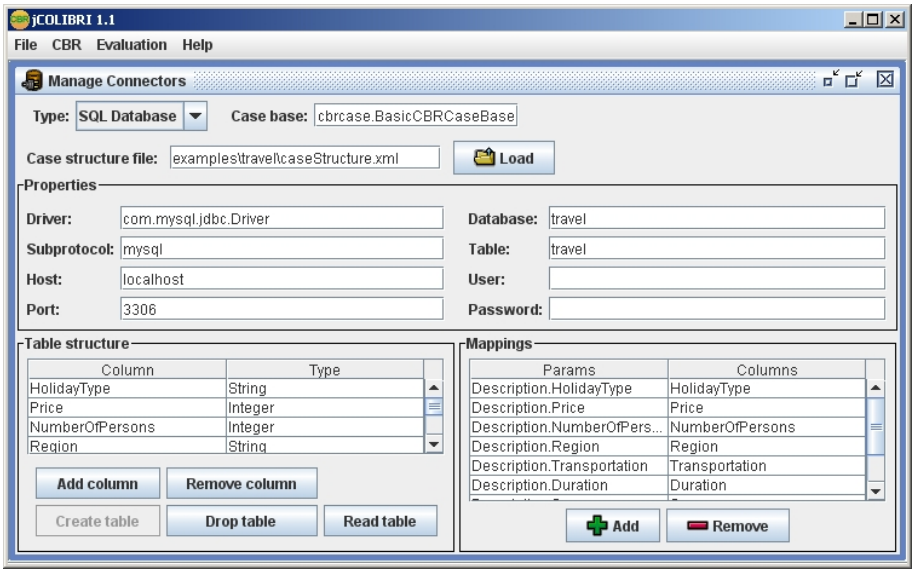


Figura 3.6: GUI de configuración de los conectores

ponentes del caso (descripción, solución, y resultado) y permite añadir atributos a cada uno de estos componentes. Por cada atributo pueden especificarse el tipo de datos almacenado y la función de similitud que se utilizará en la recuperación.

Esta versión del armazón esta muy ligada al método de recuperación k-NN y por lo tanto permite configurarlo gráficamente. Como se explicó en la Sección 2.1.1, el cálculo del vecino más cercano suele realizarse mediante funciones de similitud locales que obtienen la similitud de cada atributo, y funciones de similitud globales que suelen realizar una media ponderada con los valores devueltos por las medidas locales. Para poder configurar este proceso gráficamente, la interfaz encargada de gestionar la estructura del caso permite asignar funciones de similitud a cada uno de los atributos. Además se permite asignar un peso a cada atributo y fijar los parámetros de las medidas de similitud tal y como se muestra en la Figura 3.7.

- Configuración de las medidas de similitud. Esta interfaz permite definir las distintas medidas de similitud incluidas en el armazón y especificar los tipos de datos sobre los que puede aplicarse cada una de ellas. Esto se debe a que cada función debe establecer sobre qué tipos trabaja ya que, por ejemplo, no es lo mismo una medida que trabaje sobre cadenas de texto o sobre números decimales.

- Configuración de los tipos de datos. jCOLIBRI 1 requiere una definición

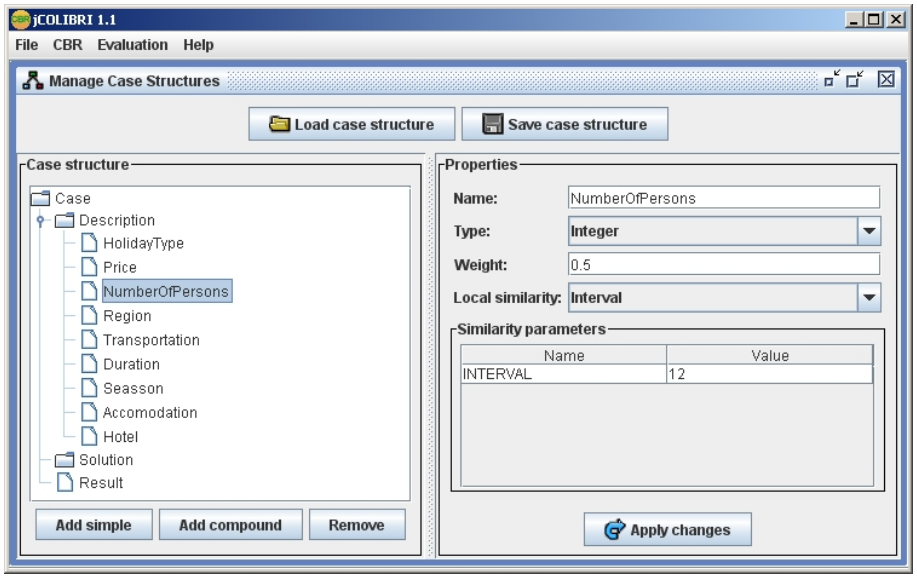


Figura 3.7: GUI de configuración de la estructura del caso

especial de los tipos de datos que establecen los valores asignables a cada atributo. Gracias a ello, es posible conocer cómo deben ser serializados al guardarlos en el medio de persistencia, cómo se han de generar los cuadros de diálogo que permiten obtener sus valores y qué medidas de similitud son aplicables. Por lo tanto, la interfaz de configuración de los tipos de datos permite realizar toda esta configuración de forma gráfica.

- Obtención y modificación de la consulta y casos. Los archivos de configuración generados a través de las herramientas de configuración de la estructura del caso y tipos de datos permiten crear automáticamente cuadros de diálogo para la edición de los valores de la consulta o los casos. Mediante la estructura del caso es posible obtener qué tipos de datos tiene cada atributo, y en la información sobre dichos tipos se añade qué tipo de elemento gráfico permite su obtención y edición. De esta forma algunos métodos del armazón pueden generar cuadros de diálogo que posibilitan al usuario introducir la consulta o modificar los valores de los atributos durante la etapa de adaptación.

3.1.2.3 Ejecución dinámica y generación del código

La herramienta de composición de métodos permite asignar a cada una de las tareas un método que las resuelve. Durante este proceso se crean instanciaciones de los métodos donde se ha añadido información sobre los parámetros

necesarios para su ejecución. Gracias a esta información se posibilita la ejecución dinámica de los métodos mientras se esta realizando este proceso de configuración. Esta funcionalidad se engloba en una herramienta que permite ejecutar un conjunto de métodos mediante la pulsación de un simple botón. Esta característica permite probar cada uno de los métodos implementados (cuando no dependan de los anteriores) o un conjunto de métodos aislado (por ejemplo los métodos de recuperación).

Una vez que el proceso de configuración ha terminado, la interfaz gráfica de jCOLIBRI 1 permite generar el código fuente java que contiene la funcionalidad diseñada. Para realizar esta tarea se utilizan una serie de plantillas que contienen el esqueleto de una aplicación CBR diseñada con jCOLIBRI. Estas plantillas de código contienen las declaraciones de los métodos preciclo, ciclo y postciclo. A continuación estos métodos son rellenados con la configuración definida por el usuario a través de las herramientas visuales.

De esta forma se obtiene el código de una aplicación CBR completamente funcional e independiente de la herramienta de configuración (aunque obviamente siempre se dependerá de las librerías de jCOLIBRI). Este proceso posibilita que el sistema CBR generado pueda ser extendido o integrado en otras aplicaciones.

Además, jCOLIBRI 1 permite compilar y ejecutar todo el código generado a través de la interfaz gráfica. Así se evita la invocación del compilador y la máquina virtual de java externamente. Esto permite comprobar directamente el funcionamiento de la aplicación CBR final según se muestra en la Figura 3.8.

Como ya se observó en la Sección 2.4.2 esta característica es típica de los denominados "generadores de aplicaciones" y no de los armazones. Sin embargo, en jCOLIBRI 1 se incluye esta funcionalidad porque aumenta enormemente la versatilidad y utilidad de la plataforma.

3.1.2.4 Evaluación de aplicaciones CBR

Mediante las distintas herramientas descritas en el apartado anterior, el desarrollo de sistemas CBR con jCOLIBRI se realiza de forma rápida y sencilla. Esta facilidad de desarrollo extiende el uso del armazón para la enseñanza de sistemas CBR y el prototipado rápido de aplicaciones experimentales. Por lo tanto, se hace necesario un mecanismo que permita a las distintos usuarios (estudiantes, investigadores, desarrolladores, etc.) evaluar las aplicaciones generadas. Para satisfacer esta necesidad, jCOLIBRI 1 incorpora una serie de métodos que permiten evaluar fácilmente los sistemas CBR generados con la herramienta.

Durante el diseño de esta extensión ya se tienen en cuenta los problemas

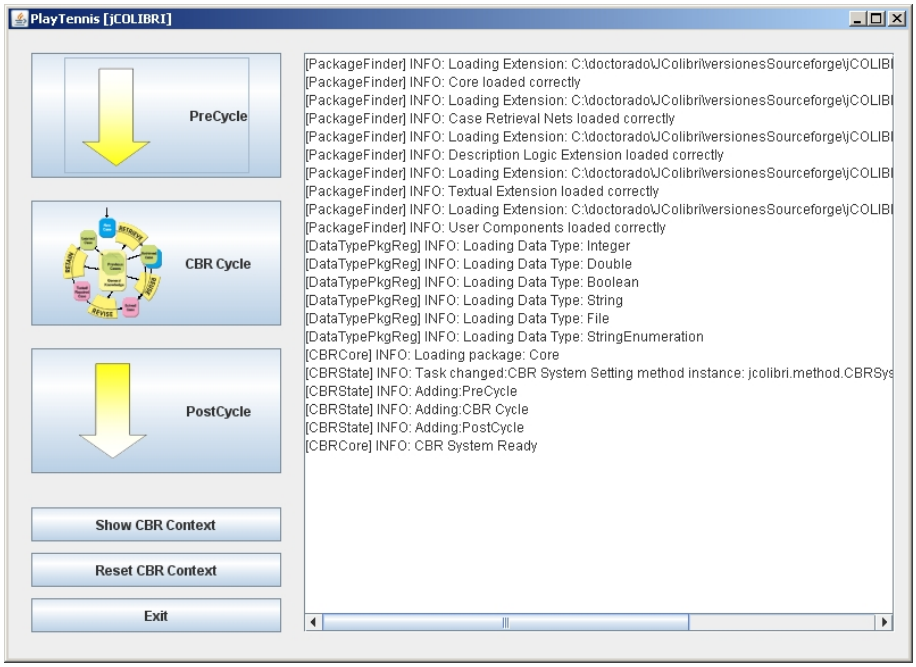


Figura 3.8: Aplicación CBR independiente generada por jCOLIBRI 1

de acoplamiento entre el código y herramientas visuales que desembocarán en la futura arquitectura de jCOLIBRI 2 y que serán detallados en la Sección 3.1.3. Para evitar este inconveniente se desarrolla una extensión donde el código de evaluación y las herramientas gráficas que lo configuran están totalmente desacoplados. Este diseño permitió la reutilización directa de toda la extensión en las siguientes versiones del armazón.

Los métodos de evaluación desarrollados son:

- Leave One Out. Consiste en extraer un caso de la base de casos y utilizarlo como consulta.
- Hold Out. Divide la base de casos en dos conjuntos: casos utilizados como consultas y casos que forman la base de casos en cada evaluación.
- N-Fold. Divide la base de casos en varios conjuntos. En las distintas iteraciones de la evaluación un conjunto de casos se utiliza como consultas y los restantes como base de casos.

En el diseño de los métodos de evaluación se definen con claridad las interfaces necesarias para que cualquier desarrollador pueda implementar los suyos propios cuando las tres implementaciones anteriores no son suficientes.

En la figura 3.9 se muestra la herramienta de configuración de la evaluación y el resultado de la misma. En este ejemplo se está utilizando el

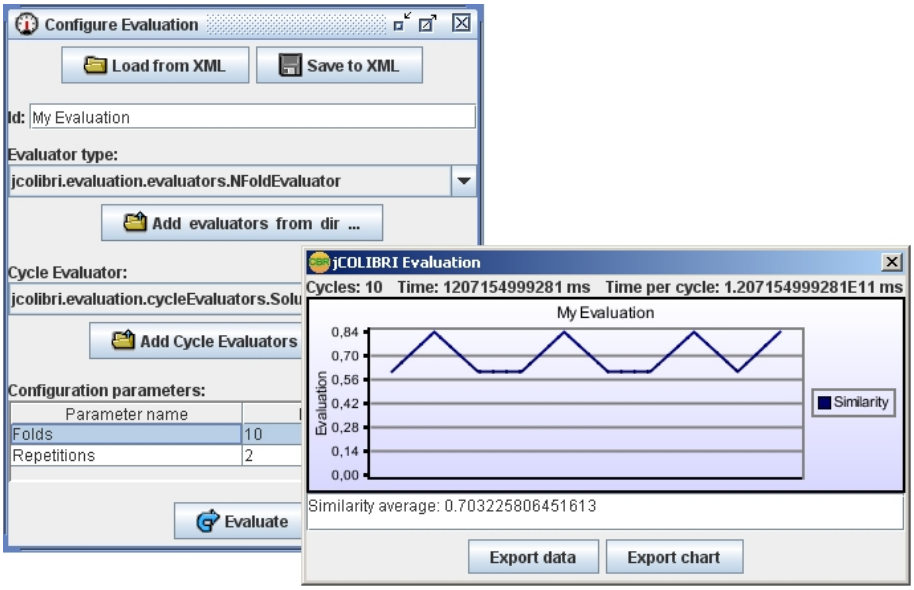


Figura 3.9: GUI de evaluación de sistemas CBR

método N-Fold para evaluar el valor de similitud de los casos. La interfaz permite establecer los parámetros del método y guardar en archivos XML toda la configuración de la evaluación. De esta manera se posibilita reproducir exactamente el mismo experimento sin tener que repetir el proceso de configuración. Una vez finalizada la evaluación se genera automáticamente una gráfica con los resultados del experimento. Estos resultados pueden ser exportados a distintos formatos de archivo, tanto gráficos (jpg) como de datos (csv).

3.1.2.5 Integración en aplicaciones web

Aunque la integración en aplicaciones web no fue uno de los objetivos principales de jCOLIBRI 1, en esta versión se tienen en cuenta la importancia de estas tecnologías para la implantación de los sistemas CBR generados. La creciente popularidad de las aplicaciones web y la especial adecuación del paradigma CBR a procesos como la recomendación de productos convierten la compatibilidad web de jCOLIBRI en un requisito prioritario.

Consecuentemente, el esta versión se incorpora una extensión que posibilita la integración de las aplicaciones generadas con jCOLIBRI en servidores web Java (como por ejemplo, Apache). Sin embargo, esta integración no resulta demasiado "natural" debido a la inversión del flujo de control presente de los armazones. En la Sección 2.4.1 se comentó el denominado *principio de Hollywood* que representa este control del flujo de la aplicación por parte de

los armazones. En esa sección también se detallaron los problemas que este principio presenta para la integración de armazones dentro de aplicaciones web, ya que en este tipo de aplicaciones la inversión del control no es una característica viable.

jCOLIBRI 1 también tiene esta característica de los armazones y por lo tanto controla el flujo de llamadas a los métodos. Esto supone un grave problema y obliga a que la extensión implementada realice las llamadas a las páginas del servidor pero siempre manteniendo el control de esas invocaciones desde jCOLIBRI. Obviamente esto presenta varios inconvenientes para el desarrollo de aplicaciones CBR con interfaz web y es una de las razones que desencadenan en la nueva arquitectura de jCOLIBRI 2.

3.1.2.6 Documentación

La calidad de la documentación es uno de los factores mas importantes para la popularización de un armazón. En la Sección 2.4.1 (pág. 59) se exponían los principales tipos de documentación junto con sus ventajas e inconvenientes. En esa revisión se planteaban tres tipos: manuales de referencia, tutoriales y aplicaciones de ejemplo.

En jCOLIBRI 1 se incluye un manual de referencia con la documentación de la interfaz de programación. Esta documentación se genera automáticamente mediante de la herramienta JavaDoc a partir de los numerosos comentarios incluidos en el código fuente.

Además, en vez de crear tutoriales y aplicaciones de ejemplo por separado, se proporcionan una serie de video-tutoriales que ofrecen una visita guiada por las distintas capacidades del armazón según se van implementando distintos tipos de aplicaciones. De esta forma se combinan las características de ambos tipos de documentación: se ofrece una aproximación guiada al uso de la herramienta y además se enseña cómo implementar aplicaciones de complejidad cada vez más creciente. Obviamente este tipo de tutorial sólo es posible debido a las numerosas interfaces gráficas incluidas en jCOLIBRI 1 que permiten la configuración completa de aplicaciones CBR sin la necesidad de implementar complejos fragmentos de código.

Toda esta documentación se ofrece a través del sitio web de jCOLIBRI [3] junto con una introducción a la arquitectura y diversos diagramas UML de clases y secuencia.

3.1.3 Problemas de la plataforma

La primera versión de jCOLIBRI incluye una gran cantidad de funcionalidad que es fácilmente utilizable mediante las interfaces gráficas ofrecidas.

[3] http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri1/index.html

Esta aproximación es muy útil para diseñadores de aplicaciones CBR que no quieren entrar en los detalles del código y prefieren prototipar el sistema rápidamente. Sin embargo, esta desviación hacia los usuarios con perfil diseñador es contraproducente para los usuarios con un perfil más orientado a desarrollador ya que este tipo de usuarios prefiere controlar los detalles de la aplicación mediante el código fuente.

Como se explicaba en la sección 2.4.2, existen dos tipos de armazones:

- Los armazones de *caja blanca* están diseñados para que el desarrollador extienda las clases e interfaces del código. Por lo tanto, el desarrollador conoce los detalles de funcionamiento interno de los componentes del armazón.

- En el extremo contrario, los armazones de caja negra consisten en componentes y clases que son instanciados y configurados por los desarrolladores. Normalmente, los componentes de una armazones de caja negra se basan en elementos de armazones de caja blanca. La composición y configuración de los elementos en los armazones de caja negra se realiza mediante herramientas de configuración y resulta bastante más sencilla que la instanciación de los elementos de los armazones de caja blanca.

A partir de estas definiciones, está claro que jCOLIBRI 1 es un armazón de caja negra. La principal ventaja de este tipo de armazones es la facilidad de uso y esto constituye una de las principales virtudes de esta versión. Sin embargo hay que tener en cuenta una premisa muy importante: un armazón de caja negra tiene que estar basado en un buen diseño de caja blanca. Si no es así, las modificación y extensión de los componentes de caja negra se vuelve progresivamente más complicada. Además, es bastante probable que los componentes de caja negra no sean suficientes para el desarrollador en algún tipo de aplicación. Por ello, hay que permitir que el programador pueda "bajar" al nivel de caja blanca para utilizar el código del armazón con más control y detalle. Si la parte de caja negra no está fundada en un buen diseño de caja blanca los desarrolladores no pueden manejar el código del armazón de forma sencilla, lo que supone un gran problema de diseño.

Dichas deficiencias están presentes en jCOLIBRI 1. Esta versión es un buen armazón de caja negra, pero carece de una parte de caja blanca clara y bien diseñada. Esto originaba problemas para desarrolladores a quienes las interfaces gráficas no eran suficientes y necesitaban bajar a nivel de código para implementar sus aplicaciones.

Un claro ejemplo de este problema es el código generado por la interfaz. jCOLIBRI 1 permite generar el código fuente de la aplicación CBR una vez configurados los métodos que resuelven cada tarea. Sin embargo, al observar el código generado para la invocación de un método nos encontramos el código del listado 3.2:

Listado 3.2: Ejemplo de código de caja negra

```
1  task = TasksInstanceRegistry.getInstance().createInstance("Select␣best
       ␣task","Select␣best␣task4895");
2  taskList.add(task);
3  method = MethodsInstanceRegistry.getInstance().createInstance("
       jcolibri.method.SelectSomeCasesMethod","jcolibri.method.
       SelectSomeCasesMethod9091");
4  core.resolveTaskWithMethod(task, method);
5  //These are serialized version of the values
6  //assigned during application configuration.
7  parametersMap = new HashMap<String,Object>();
8  //parameter:Number of cases=3
9  parametersMap.put("Number␣of␣cases", jcolibri.util.JColibriClassHelper
       .deserializeObject(URLDecoder.decode("EF␣BE␣AC␣EF␣BF␣AD␣00␣05sr␣00
       ␣11java.lang.Integer␣12␣EF␣BF␣A2␣EF␣BE␣A0␣EF␣BE␣A4␣EF␣BF␣B7␣EF␣BE␣
       81␣EF␣BE␣878␣02␣00␣01I␣00␣05valuexr␣00␣10java.lang.Number␣EF␣BE␣86
       ␣EF␣BE␣AC␣EF␣BE␣95␣1D␣0B␣EF␣BE␣94␣EF␣BF␣A0␣EF␣BE␣8B␣02␣00␣00xp␣00␣
       00␣00␣03","UTF-8")));
10 method.setParameters(parametersMap);
```

Este es el código resultante de instanciar el método "Select Best Cases". La figura 3.10 muestra cómo se realiza gráficamente este proceso. El usuario ha seleccionado la tarea "Select Best Task" y el sistema ha generado una instancia de esa tarea llamada "Select best task4895" (en el listado 3.2 aparece en la línea 1). Consiguientemente, la interfaz habrá mostrado en el panel "Methods" los métodos aplicables para resolver dicha tarea. En este caso, el usuario elige el método "Select Best Cases" y al pulsar sobre el botón de instanciación se genera una instancia de ese método llamada "jcolibri.method.SelectSomeCasesMethod9091" (línea 3 del listado). A continuación, hay que configurar los parámetros del método. En este caso se pregunta por el número de casos a recuperar. Este parámetro es de tipo Integer y la GUI genera automáticamente una ventana de dialogo que obtiene este valor. Como estamos ante un armazón de caja negra, no importan los detalles de cómo se instancia la clase, lo que importa es obtener un componente configurado y listo para ejecutar. Por lo tanto el código generado no se encarga de crear variables que contengan esos parámetros ya que requiere un análisis muy detallado del código y no permitiría que la GUI manejara métodos ya instanciados. En su lugar se serializa el contenido de esos parámetros en una cadena. Esta cadena sí que es manejable fácilmente por la GUI, pero al generar el código hay que decodificar esa cadena para poder asignar los parámetros al método correspondiente (línea 9 del listado).

Obviamente, esta forma e instanciar los métodos puede resultar útil para su gestión desde la interfaz gráfica. Sin embargo, desde el punto de vista del programador es un auténtico problema ya que es imposible seguir el flujo del programa. En el listado 3.3, se muestra el código de caja blanca que se utiliza en jCOLIBRI 2 para realizar esta misma acción:

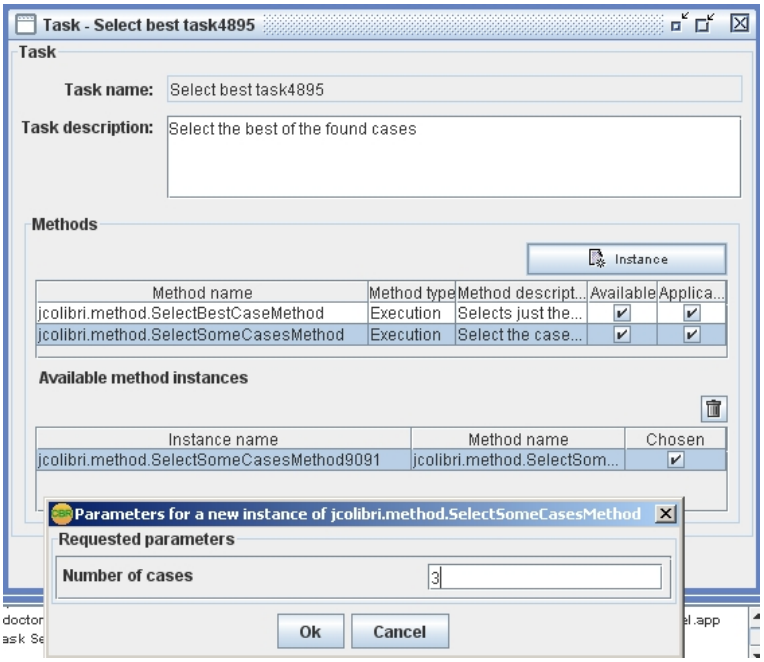


Figura 3.10: Instanciación de métodos en jCOLIBRI 1

Listado 3.3: Ejemplo de código de caja blanca

```
// Select k cases
eval = SelectCases.selectTopKRR(eval, 3);
```

Por lo tanto, desde del punto de vista del programador es muchísimo más sencillo utilizar las clases directamente que reutilizar el código de los componentes instanciados en un armazón de caja negra.

Aunque el principal problema de jCOLIBRI 1 es la instanciación de sus componentes, existen otros problemas de diseño. La mayoría de estos problemas se basan en decisiones de diseño que se tomaron en las etapas iniciales del armazón. Aunque estas decisiones parecieran acertadas en los primeros compases de desarrollo, la progresiva expansión del armazón demostró que algunas no escalaban adecuadamente según se ampliaba la funcionalidad. El siguiente listado muestra las principales dificultades en el diseño del armazón que se encontraron durante su vida útil:

Representación de los casos. El diseño de clases que representa la estructura de los casos mezcla datos y metadatos. Por datos nos referimos a los valores de cada atributo, mientras que por metadatos nos referimos a información sobre la similitud o peso de los atributos que será utilizada por algoritmos como el k-NN. En las etapas iniciales del

armazón, incluir la información de similitud en la estructura del caso no era un mal diseño ya que sólo se disponía del método k-NN. Sin embargo, en futuras versiones se añadieron métodos que no utilizan dichos datos (recuperación en ontologías, mediante filtros, etc.).

Además no tiene demasiado sentido que cada caso en la base de casos almacene información sobre similitud. Dicha información debería poder ser configurada al introducir la consulta y ser independiente de los casos cargados en memoria.

Otro problema muy importante es que el patrón *Composite* utilizado para representar la estructura de los casos (Figura 3.4) dificulta mucho la depuración. Al no contener los valores de los atributos directamente como variables del objeto hay que seguir las relaciones con otros individuos para conocer dichos valores. Al depurar aplicaciones CBR es muy útil poder acceder fácilmente a los valores de los atributos de los casos y esta representación no lo permite.

Metadatos. Además del problema con los metadatos de similitud en la representación de los casos existen redundancias en la información almacenada en ficheros XML de configuración. Esta información podría ser inferida directamente del código o de los objetos en ejecución. Por ejemplo, los parámetros de los métodos se especifican en un fichero XML como el mostrado en el listado 3.1. Esta información se podría inferir fácilmente de la signatura de los métodos una vez hayan sido cargados por la máquina virtual de Java. Otro ejemplo, es la representación en XML de la estructura de los casos. Aunque esto no es un inconveniente real, sí es un problema de redundancia que perjudica la depuración y mantenimiento del software.

Arquitectura de pizarra. Al describir la arquitectura del armazón se explicó que los métodos utilizaban una clase denominada `Contexto` para intercambiar la información. Este mecanismo se denomina comúnmente como arquitectura de pizarra, y es una manera simple y extensible de permitir la comunicación entre componentes. Sin embargo, este tipo de arquitecturas están desaconsejadas en general porque presentan más inconvenientes que ventajas cuando los sistemas software alcanzan un alto grado de complejidad. Si existen muchos componentes leyendo y escribiendo datos en un pizarra resulta muy difícil controlar los identificadores utilizados para etiquetar los datos leídos y producidos. Además, este tipo de diseño no permite garantizar el orden de ejecución de los métodos.

Este problema se acentúa aún más cuando la aplicación se compone de diferentes hebras de ejecución. Normalmente, las aplicaciones CBR no utilizan varias hebras y el diseño de jCOLIBRI 1 se basa en esa premisa. Sin embargo, durante la evolución del armazón se encontraron

varios escenarios donde sí era necesario utilizar hebras. El ejemplo más claro es la interfaz web, ya que permite que varios usuarios ejecuten la aplicación CBR de forma concurrente.

Expresividad de la consulta. En jCOLIBRI 1 las consultas son casos sin solución. Por lo tanto el usuario puede rellenar los atributos de una consulta y utilizar esos valores para recuperar los casos más similares. De esta forma se obtienen los casos cuyos atributos son similares a los atributos de la consulta. Este mecanismo implica que el sistema aplique el operador booleano Y en todos los atributos:

$$Caso \approx Q \rightarrow Caso_{at1} \approx Q_{at1} \wedge Caso_{at2} \approx Q_{at2} \wedge \ldots$$

Sin embargo esta interpretación puede no ser suficiente. Supongamos la siguiente consulta en un sistema CBR sobre coches: "Quiero un coche diésel, de 3 puertas y color rojo, ...". Esta consulta puede ser expresada con el lenguaje actual ya que se compone de tres atributos evaluados con la función Y. Pero imaginemos la siguiente consulta: "Quiero un coche diésel o gasolina, no me importa el número de puertas y prefiero cualquier color que no sea el verde". Aquí el usuario está utilizando expresiones más complejas como NO, O, CUALQUIERA, NINGÚN. Además, podríamos definir la consulta de forma todavía más compleja si se permitieran comparaciones numéricas, por ejemplo: "Menos de 5000 km.".

Con estos ejemplos se pone de manifiesto que el mecanismo de representación y obtención de la consulta a través de un cuadro de diálogo que permite introducir los valores de cada atributo limita la expresividad del usuario. Por lo tanto, sería deseable ampliar este esquema para permitir consultas más complejas.

Flujo de control. Normalmente los armazones llevan el control del flujo de la aplicación y llaman a los componentes desarrollados por el usuario. Este concepto se denomina *inversión de control* o *Principio de Hollywood* (Sparks et al., 1996) y es opuesto al diseño de las librerías de componentes donde se ofrecen distintos métodos que serán invocados por el desarrollador.

En principio jCOLIBRI 1 es un armazón con inversión de control ya que se encarga de invocar a los distintos métodos que han sido configurados por el usuario para el preciclo, ciclo y postciclo. Sin embargo, que jCOLIBRI siga únicamente esta aproximación es un problema porque perjudica su integración en otros sistemas que simplemente necesitan invocar a los métodos CBR pero manteniendo el control de flujo. El ejemplo más obvio es la integración en servidores web. Por lo tanto, jCOLIBRI debe proporcionar un control de flujo pero a su vez permitir invocaciones externas a sus métodos.

Representación de las pre-/postcondiciones. Aunque el fichero de configuración de los métodos reserva un campo para las pre-/postcondiciones nunca se llegó a definir un mecanismo para expresarlas. Estas condiciones no son imprescindibles para la configuración de una aplicación CBR pero con el aumento del número de métodos son muy necesarias. Para ilustrar esta idea supongamos una aplicación donde el diseñador haya seleccionado un método de CBR textual para la recuperación y a continuación un método de adaptación con ontologías. Aunque esta situación sería teóricamente posible, no tiene demasiado sentido si no se utiliza una representación de casos y atributos muy concreta. Para evitar este tipo de situaciones, las pre-/postcondiciones de los métodos deberían incluir información sobre estas limitaciones de composición junto con los datos necesarios para su ejecución (que deberían estar presentes en la pizarra antes de invocar al método).

Ejecución secuencial de los métodos. El proceso de descomposición sucesiva en subtareas únicamente permite diseñar aplicaciones donde los métodos se ejecutan de forma secuencial. Sin embargo, en muchas ocasiones este tipo de ejecución es insuficiente al necesitarse construcciones más complejas como condicionales y bucles. Un ejemplo de este requisito lo encontramos en los sistemas CBR conversacionales o de recomendación donde se necesitan sucesivas iteraciones para refinar los requisitos del usuario.

Los problemas de la arquitectura de jCOLIBRI 1 expuestos en esta sección resultaron ser muy útiles para la evolución general del armazón aunque en principio pudieran verse como algo negativo. A partir de las conclusiones obtenidas de su análisis, se diseñó la nueva arquitectura de jCOLIBRI 2 expuesta a continuación. De esta forma, el diseño de la segunda versión carece de todos estos inconvenientes y es mucho más versátil, extensible y fácil de utilizar. Toda esta experiencia en el desarrollo de armazones CBR (problemas potenciales y posibles soluciones) fue presentada en Recio-García et al. (2006b).

3.2 jCOLIBRI 2

jCOLIBRI 2 es el resultado de la experiencia adquirida con el desarrollo de la primera versión. En esta nueva implementación se resuelven los problemas aparecidos en su predecesora como la representación de los casos, la gestión de los metadatos, la complejidad del código generado, etc. La principal diferencia entre las dos versiones radica en el tipo de armazón. Según se comentó en la Sección 3.1, jCOLIBRI 1 era un buen armazón de caja negra formado por interfaces gráficas que permitían la composición de los distintos métodos

que incorporaba. Sin embargo, dicho armazón no estaba construido sobre los pilares de una sólida componente de caja blanca. Los armazones de caja blanca son los que se utilizan mediante extensión o herencia de las distintas clases y están orientados a usuarios desarrolladores que programan directamente el código de la aplicación. A partir de ellos se construye la parte de caja negra que permite configurar y componer las distintas partes del armazón de forma sencilla por parte de los diseñadores que no quieren entrar en los detalles del código. La carencia de este buen diseño de caja blanca es el origen de jCOLIBRI 2. La segunda versión del armazón nace de la premisa de que una buena base de caja blanca posibilitaría el desarrollo de herramientas de composición de métodos como las de jCOLIBRI 1 pero sin las deficiencias encontradas en él.

En esta sección nos centraremos en describir la parte de caja blanca de jCOLIBRI 2, dejando para un capítulo aparte la componente de caja negra debido a su importancia dentro del trabajo de tesis. Por lo tanto, los siguientes apartados describirán la arquitectura e implementación de esta segunda versión realizando un breve repaso a toda la funcionalidad incorporada (esto también incluirá los métodos ya desarrollados en jCOLIBRI 1).

3.2.1 Diseño y Arquitectura

La arquitectura de jCOLIBRI 2 mantiene la doble orientación para usuarios desarrolladores y usuarios diseñadores. Esta división se muestra en la Figura 3.11 donde se distinguen claramente dos capas orientadas a usuarios desarrolladores y usuarios diseñadores. Esta característica se hereda de jCOLIBRI 1 aunque se hace una división todavía más pronunciada ya que en la versión inicial se mezclaba el código de caja blanca y caja negra. En esta sección detallaremos la parte de abajo de esta figura y dejaremos para el Capítulo 6 la descripción de la parte de arriba.

Debido a la amplia división entre ambas capas, jCOLIBRI 2 es principalmente un armazón de caja blanca al que se le añaden una serie de herramientas que permiten la composición de sus distintos componentes. Esto es posible gracias al nuevo diseño de los métodos y la corrección de los errores detallados en la sección 3.1.3, tal y como expondremos a lo largo de esta sección.

En esta capa inferior de la arquitectura destacan la nueva representación de los casos –descrita en la Sección 3.2.1.1–, junto con la librería de métodos para manejar dichos casos –que se expone en la Sección 3.2.2. Estos métodos estarán definidos en base una serie de interfaces que describen claramente los distintos componentes del armazón. Por último, estos métodos se utilizarán dentro aplicaciones CBR estructuradas en diversas etapas (preciclo, ciclo, etc.) que se presentarán en la Sección 3.2.1.2.

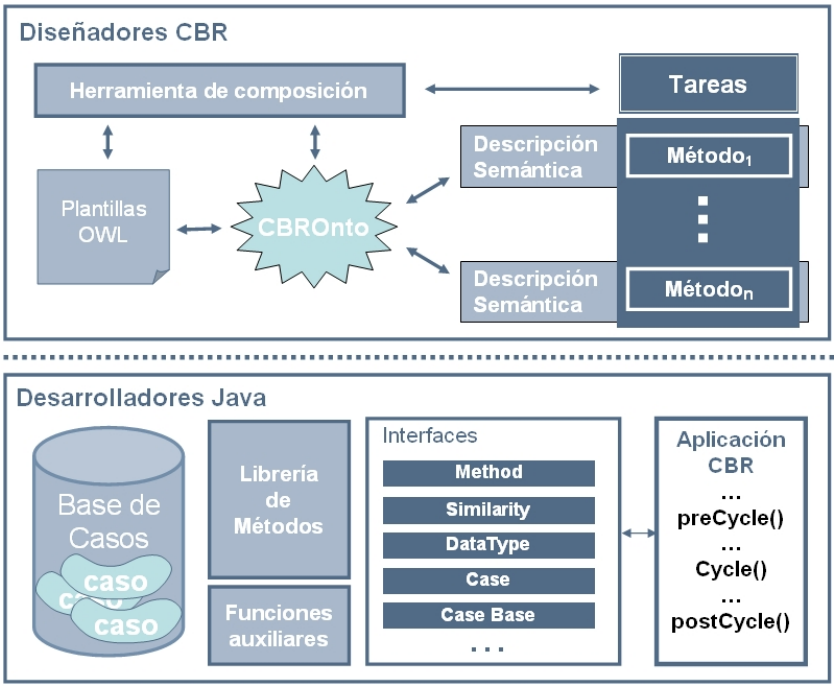


Figura 3.11: Arquitectura de jCOLIBRI 2

Antes de detallar todos estos cambios haremos un resumen de las características que se mantienen en ambas versiones del armazón:

- Diseño en capas de la arquitectura: persistencia, núcleo e interfaz de usuario. Aunque se mantiene esta organización de la arquitectura, la implementación de estas capas varía considerablemente al utilizar las tecnologías J2EE.

  La persistencia se sigue organizando mediante *conectores* que permiten acceder al medio utilizado para almacenar los casos, y diversas *organizaciones en memoria* de la base de casos. Por lo tanto, se mantienen las interfaces `CaseBase` y `Connector` de jCOLIBRI 1.

  En esta versión no se incluye ninguna implementación por defecto de la interfaz de usuario como sí ocurría en su predecesora. El motivo de esta decisión se justifica en el acoplamiento entre parte lógica y GUI que apareció en jCOLIBRI 1 al incluir esta última componente. Por lo tanto, en jCOLIBRI 2 no se incorporan clases para implementar la GUI a excepción de la extensión para recomendadores presentada en el Capítulo 6. Otro motivo que fundamenta la no incorporación de estas clases es la gran diferencia entre interfaces de usuario para aplicaciones independientes o web. Como en esta segunda versión se

presta gran atención a la integración web, la incorporación de interfaces por defecto pierde importancia.

- División de la aplicación: preciclo, ciclo y postciclo. Además se añaden una serie de posibles etapas de la aplicación que podrán ejecutarse en distintos momentos de su ciclo de ejecución. Esta aportación permite ejecutar también otras tareas como las de mantenimiento de la base de casos.

- Estructura del caso: descripción, solución, resultado y justificación de la solución. Esta última componente se añade a la estructura del caso ya que aparece en muchos trabajos encontrados en la literatura de CBR (Kolodner, 1993; Althoff et al., 1995).

La principal ventaja de conservar todos estos elementos en jCOLIBRI 2 es que facilita la transición de los usuarios entre ambas versiones del armazón.

Los siguientes apartados detallan los cambios más importantes en la arquitectura de jCOLIBRI 2 destacando las mejoras incorporadas y su implementación mediante las tecnologías J2EE.

3.2.1.1 Representación de los casos

La representación de los casos es una de las principales modificaciones de la nueva versión. En vez de mantener la estructura del patrón *Composite* presentada en la Sección 3.1.1.4, se utilizan los *Java Beans* de J2EE. Un Java Bean (o JBean) es una clase normal pero que añade métodos accedentes y mutadores (`get()` y `set()`) para cada atributo. De esta forma el desarrollador puede representar las distintas partes del caso (descripción, solución, etc.) como una clase normal donde cada atributo de la misma es uno de los atributos del caso. Sin embargo, esto presenta un problema para los métodos del armazón ya que deben acceder a atributos con tipo y nombre arbitrarios definidos en clases externas. La solución a este problema se encuentra en otra tecnología J2EE complementaria a los JBeans denominada *Introspección*. Esta tecnología permite analizar en tiempo de ejecución los atributos (nombre y tipo) de cualquier JBean y modificar dichos atributos gracias a los métodos accedentes y mutadores comentados anteriormente.

Los detalles de uso de la introspección se ocultan gracias a la nueva clase del armazón `Attribute`. Esta clase representa los atributos de cualquier JBean que se haya utilizado para representar una componente del caso. De esta forma, los métodos del armazón podrán acceder a cualquier atributo del caso y modificar sus valores fácilmente.

Cada parte del caso: descripción, solución, justificación de la solución y resultado, serán un JBean distinto creado por el desarrollador de la aplicación. Estas clases deberán implementar la interfaz `CaseComponent` para que puedan ser manejadas de forma transparente por los métodos del armazón

(ya que si no habría que parametrizar cada método con las distintas clases generadas para representar el caso). Esta interfaz únicamente define un método que devuelve el atributo (objeto `Attribute`) que hace de identificador para esa componente del caso. Este identificador servirá principalmente como clave primaria en las bases de datos, aunque de forma genérica se utilizará para indexar los casos e identificar cada una de sus componentes.

En Listado 3.4 muestra la definición de la solución de un caso y cómo se puede manipular sus atributos a través del mecanismo de introspección. Este ejemplo está tomado del tutorial del jCOLIBRI donde se implementa un recomendador de viajes (Recio-García et al., 2007b).

Listado 3.4: Ejemplo de implementación de un componente del caso

```
public class TravelSolution implements jcolibri.cbrcore.CaseComponent
    {

    String id;
    Integer price;
    String hotel;

    public Integer getPrice(){
        return price;
    }

    public void setPrice(Integer price){
        this.price = price
    }

    public Attribute getIdAttribute() {
        return new Attribute("id", this.getClass());
    }
    ...
}
...
CaseComponent solution = case.getSolution();
Attibute priceAtt = new Attribute("price", solution.getClass());
priceAtt.setValue(new Integer(2000));
```

Este componente incluye tres atributos que contienen el identificador, precio y hotel. El identificador es devuelto por el método `getIdAttribute` impuesto por la interfaz `CaseComponent`, y para todos ellos se definen métodos accedentes y mutadores. Las últimas líneas muestran cómo se puede acceder y modificar un atributo de la solución sin necesidad de mantener referencias tipadas con dicha clase (`TravelSolution`) sino a través de la interfaz `CaseComponent`.

En el ejemplo anterior se está accediendo a la solución del caso a través del método `getSolution()`. Además de este método, existirán otros tantos para acceder a los otros componentes del caso. Siguiendo el diseño de jCOLI-

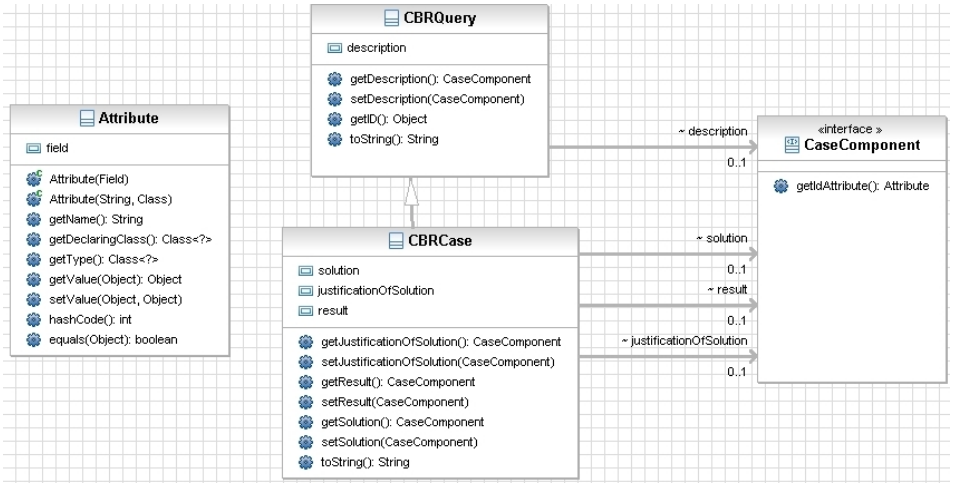


Figura 3.12: Estructura de los casos en jCOLIBRI 2

BRI 1, la clase `CBRCase` sigue heredando de la clase `CBRQuery`. En esta última se definen los métodos para manejar la descripción del caso y en la clase `CBRCase` se añaden los métodos para manipular la solución, la justificación de la solución y el resultado. Figura 3.12 muestra el diagrama de clases completo que define toda la estructura para la representación de los casos.

Este nuevo diseño para la representación de los casos soluciona dos de los problemas encontrados en la versión inicial:

1. Se facilita la representación y depuración de los casos. la representación de los casos es mucho más intuitiva al no tener que mantener la estructura de individuos relacionados con otros individuos. Además, se facilita enormemente la depuración de las aplicaciones ya que es posible acceder fácilmente a los atributos de los casos. Este último factor tiene mucha importancia para lograr la aceptación del armazón por parte de los usuarios. Este hecho se demuestra en las encuestas presentadas en el Capítulo 7 y detalladas en el Apéndice A.

2. Se mejora considerablemente la integración de jCOLIBRI en aplicaciones web ya que las tecnologías Java destinadas a implementar este tipo de interfaces se fundamentan en los JBeans. Este es el caso de tecnologías como Servlets o JSPs donde los JBeans almacenan la información obtenida y presentada desde/al usuario.

3.2.1.2 Estructuración de las aplicaciones CBR

En el diseño de esta versión se tienen en cuenta las problemáticas surgidas a partir de la inversión del control (*Principio de Hollywood*) aparecidas en

jCOLIBRI 1 que se comentaron en la Sección 3.1.3 (pág. 91. Por ello, se ofrece la posibilidad de utilizar o no esta característica.

Si se desea mantener el control sobre el flujo de la aplicación pueden utilizarse los métodos del armazón directamente. De este modo es posible implementar fácilmente aplicaciones web que incorporen la funcionalidad de jCOLIBRI 2 y el armazón se convierte en una librería de métodos ya que cumple las características que se presentaron en la Sección 2.4.2.

La opción opuesta consiste en dejar el control de la aplicación en manos del armazón por medio de la interfaz `StandardCBRApplication`. Esta interfaz se muestra en el Listado 3.5 y divide las aplicaciones CBR en cuatro etapas: configuración, preciclo, ciclo y postciclo. Las tres últimas ya aparecían en la primera versión de jCOLIBRI, mientras que la etapa de configuración se añade para realizar toda la inicialización de los métodos y componentes que se utilizarán en la aplicación (conectores, medidas de similitud, etc.).

Listado 3.5: Interfaz StandardCBRApplication

```
public interface StandardCBRApplication
{
    public void configure() throws ExecutionException;
    public CBRCaseBase preCycle() throws ExecutionException;
    public void cycle(CBRQuery query) throws ExecutionException;
    public void postCycle() throws ExecutionException;
}
```

Esta interfaz permite que los componentes del armazón puedan invocar cada una de las etapas de la aplicación y, por lo tanto, mantener el control sobre su flujo de ejecución. Esto presenta numerosas ventajas entre las que destacan la posibilidad de aplicar los métodos de evaluación de las aplicaciones o de visualización de la base de casos. Además, esta interfaz puede ser extendida mediante herencia para ajustarla a futuras necesidades.

3.2.1.3 Soluciones de los problemas de jCOLIBRI 1

En la Sección 3.1.3 se ofrecía una recopilación de los problemas en la plataforma jCOLIBRI 1 que condujeron al desarrollo de esta segunda versión. Una vez realizada la exposición sobre el diseño y arquitectura de jCOLIBRI 2 podemos detallar cómo se solucionaron los problemas aparecidos en su predecesora:

- *Acoplamiento entre métodos e interfaz gráfica.* Este problema se resuelve dividiendo la arquitectura del armazón en dos capas bien diferenciadas. La primera es una armazón de caja blanca utilizado por los usuarios desarrolladores. La otra capa está dirigida a los usuarios

diseñadores y contiene herramientas gráficas para la composición de las aplicaciones CBR sin necesidad de programar directamente sobre el código fuente. Esta última capa se detallará en el Capítulo 6.

- Los problemas en la *representación de los casos* se resuelven gracias a la utilización de Java Beans que facilitan su persistencia y depuración.

- La *redundancia de los metadatos* desaparece al utilizar la tecnología de *Introspección* para obtener en tiempo de ejecución las características sobre los casos y métodos utilizados en la aplicación.

- Arquitectura de pizarra. Esta restricción en el diseño de los métodos desaparece en la capa de caja blanca. Ahora se ofrece total libertad en la definición de la signatura de los métodos en vez de recibir la pizarra como único parámetro y obtener desde ese objeto los datos necesarios. De esta forma los parámetros de entrada y salida en los métodos de jCOLIBRI 2 se definen de forma completamente natural. Obviamente, esta libertad es contraproducente para realizar la composición semiautomática de los métodos, ya que es necesario conocer qué representan y qué tipo tienen cada uno de los parámetros de entrada/salida de un método para poder componerlo con otro. Sin embargo, esta composición de los métodos del armazón sigue siendo posible gracias a la tecnología de Introspección y la representación semántica del funcionamiento de los métodos mediante ontologías que se presentará en el Capítulo 6. En esta representación semántica del funcionamiento de los métodos se utilizarán las tecnologías desarrolladas en el área de los Servicios Web Semánticos por lo que también se resuelven los problemas de representación de las pre-/postcondiciones de jCOLIBRI 1.

- Expresividad de la consulta. Aunque este aspecto no ha sido implementado, sí que queda reflejado en la arquitectura de jCOLIBRI 2 y será incluido en futuras versiones. Para aumentar la expresividad de la consulta se propone utilizar el lenguaje HQL de Hibernate[4]. Este lenguaje es muy similar a SQL y permitiría realizar consultas complejas sobre la base de casos. El uso de Hibernate dentro del armazón se presentará en la próxima sección.

- Flujo de control. Este aspecto ya se ha explicado en la Sección 3.2.1.2 donde se plantea la posibilidad de utilizar jCOLIBRI 2 como un armazón tradicional con inversión de control o como una librería de métodos que permita mantener el flujo de control por parte del desarrollador.

- El problema de la ejecución secuencial de los métodos también desaparece completamente en esta versión. Como la componente de caja

[4] http://www.hibernate.org/hib_docs/reference/en/html/queryhql.html

blanca permite que el desarrollador implemente sus propias aplicaciones realizando llamadas a los métodos, el flujo de ejecución quedará definido por el desarrollador mediante las distintas primitivas de Java (if, while,...). Sin embargo este problema sigue estando potencialmente presente en la composición de aplicaciones mediante interfaces gráficas de la capa de diseñadores. Para resolverlo, en el Capítulo 6 se propone un sistema de composición basado en plantillas que permiten utilizar flujos de ejecución más complejos. La representación de estas plantillas se realizará mediante una ontología que contendrá elementos similares a las estructuras de control de los procesos complejos en OWL-S (ver Sección 2.3.2.1, pág. 49).

3.2.2 Funcionalidad

Una vez descritos los cambios más importantes en la arquitectura del armazón y algunos detalles sobre la implementación, queda presentar la funcionalidad incluida en jCOLIBRI que está representada como "Librería de métodos" en la Figura 3.11. En esta recopilación de los métodos del armazón también incluiremos la funcionalidad que ya estaba implementada en jCOLIBRI 1 y así ofrecer una visión global de toda la funcionalidad de nuestra plataforma.

Los métodos y componentes de jCOLIBRI 2 pueden organizarse según aparecen en la Figura 3.13. Esta figura se utiliza como punto de entrada a la documentación del armazón en la página web[5] y permite explorar las distintas partes que forman jCOLIBRI:

Capa de persistencia. Incluye los conectores que permiten acceder a al medio que almacena los casos. jCOLIBRI incluye un conector con ontologías, ficheros textuales y bases de datos. Todos ellos se configuran a través de archivos XML. El conector con ontologías será detallado en el Capítulo 5, el de ficheros textuales se hereda de la primera versión y se obtiene los casos desde las diferentes líneas de un fichero de texto plano. Por último, el conector con bases de datos está implementado mediante la librería Hibernate[6]. Esta librería es muy utilizada para implementar los servidores Web J2EE y es capaz de gestionar automáticamente la persistencia de JBeans en cualquier SGBD. La utilización de Hibernate en el conector de bases de datos aumenta enormemente su versatilidad y compatibilidad. Para ello, únicamente es necesario crear dos ficheros XML: uno con las características del SGBD y otro con el mapeo entre los atributos del caso y las distintas columnas de la base de datos. Una vez definidos dichos archivos, el conector delega en esta librería toda la gestión de la base de datos que contiene los casos.

[5] http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2/docs.html
[6] http://www.hibernate.org/

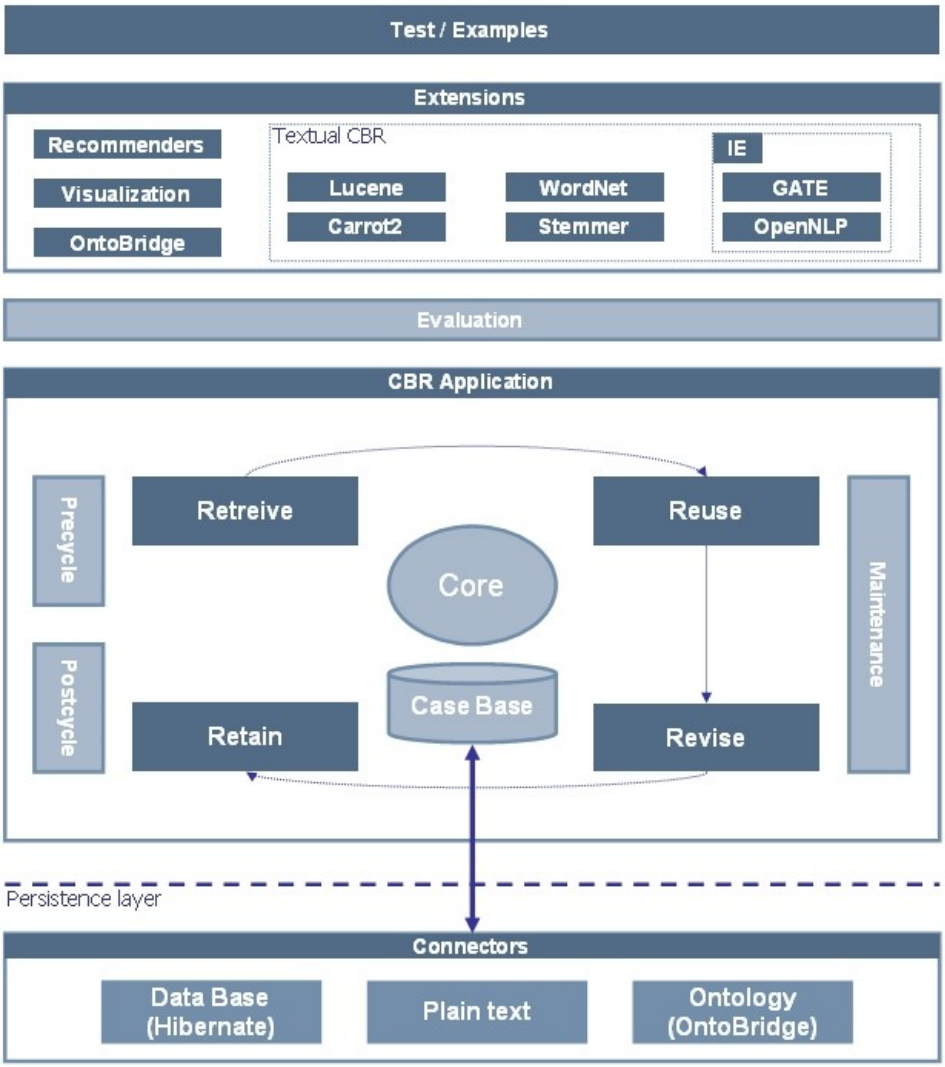


Figura 3.13: Organización de la funcionalidad de jCOLIBRI 2

La inclusión de Hibernate dentro de jCOLIBRI supone delegar en una librería muy potente, flexible y extendida una parte importante de la funcionalidad necesaria para implementar sistemas CBR.

Nucleo. Esta parte engloba las clases e interfaces más importantes del armazón. Los distintos métodos que se incorporen a jCOLIBRI deberán manejar estas clases por lo que podrían definirse como los pilares básicos del armazón. Entre las clases más importantes podemos citar: `Connector`, `CBRCaseBase`, `CBRQuery`, `CBRCase`, `CaseComponent` y `Attribute`.

Base de Casos. La organización en memoria de los casos se deriva de la interfaz `CBRCaseBase`. A partir de esta interfaz se implementan diversas implementaciones que utilizan diferentes estructuras para ordenar los casos. La forma de organización más básica es la de una lista plana (`LinealCaseBase`), aunque se ofrecen otras como la indexación por los identificadores del caso (`IDIndexedLinealCaseBase`).

La intefaz `CBRCaseBase` define también los métodos necesarios para almacenar nuevos casos en el medio de persistencia a través del conector. Este proceso puede realizarse de distintas maneras y aparece representado como "Retain" en la Figura 3.13.

También se incluye una versión de la base de casos que sólo repercute las modificaciones cuando se cierra la aplicación en vez de estar accediendo a la capa de persistencia cada vez que se modifica, añade o elimina uno de sus casos (`CachedLinealCaseBase`).

Métodos de recuperación. En este grupo destaca el método de recuperación por el vecino más próximo (NN, del inglés Nearest Neighbor). Este método utiliza medidas de similitud globales y locales según se explicó en la Sección 2.1.3 (pág. 20). La cantidad y deversidad de medidas incorporadas en jCOLIBRI son un factor muy importante para aumentar la flexibilidad y utilidad del armazón. Entre estas medidas encontramos las encargadas de comparar enumerados, cadenas de texto, números, etc. así como otras más especializadas que se irán presentando en los siguientes capítulos: funciones de similitud para CBR-Textual (Capítulo 4), con Ontologías (Capítulo 5) y específicas de sistemas de recomendación (Capítulo 6).

Además del NN existen otra serie de métodos de recuperación que también son específicos del tipo de aplicación CBR a implementar. Por ejemplo, se incluye el método de recuperación por diversidad basada en la media (`ExpertClerkMedianScoring`) utilizado en el sistema ExpertClerk de Shimazu (2002) y que pertenece al grupo de sistemas de recomendación . Otro método de este grupo sería el de recuperación por filtro (`FilterBasedRetrieval`) donde los casos deben cumplir una

serie de condiciones booleanas sobre sus atributos para ser recuperados. Estos dos métodos lo describiremos en el Capítulo 6. Cambiando al dominio del CBR Textual encontramos el método de recuperación de casos textuales a través del sistema Lucene (`LuceneRetrieval`) que se detallará en el Capítulo 4.

Una vez que los casos han sido recuperados por cualquiera de los métodos anteriormente citados es necesario seleccionar un subconjunto de ellos para presentarlos al usuario o adaptar sus soluciones. Por lo tanto, en el grupo de métodos de recuperación existen una serie de métodos encargados de seleccionar los casos. La aproximación más sencilla consiste en devolver los k mejores (`SelectCases`) o simplemente todos los casos recuperados. La combinación del método NN y el de selección de los k mejores da lugar al denominado método de recuperación de los k vecinos más próximos o k-NN. Aparte de este tipo de selección tan básica existen estrategias mucho más complejas que se utilizan en los sistemas de recomendación y que suelen seleccionar los casos más similares pero también diversos entre ellos: recuperación por diversidad `GreedySelection` o por satisfacción de requisitos `CompromiseDrivenSelection`. Estos métodos también se describirán en el capítulo correspondiente.

Métodos de reutilización y revisión. Estas dos etapas son muy dependientes de la aplicación concreta por lo que jCOLIBRI únicamente incluye métodos básicos para: copiar la solución de un caso a la consulta (`CombineQueryAndCasesMethod`), copiar valores desde atributos de la descripción a la solución (`DirectAttributeCopyMethod`), o calcular proporciones directas entre los atributos (`NumericDirectProportion Method`). Esta última medida puede utilizarse para realizar adaptaciones básicas como se muestra en el ejemplo de los viajes del tutorial. Si un viaje para 2 personas vale 1.500 y en la consulta se especifican 4, entonces habrá que adaptar el precio a 3.000 mediante una proporción directa entre estos valores.

Preciclo y Postciclo. Existen unos pocos métodos que deben ejecutarse exactamente antes o después del ciclo. Normalmente estos métodos se encargan de inicializar los recursos requeridos por otros métodos que se ejecutan durante el ciclo principal de la aplicación CBR.

Mantenimiento. Los métodos de mantenimiento fueron desarrollados para jCOLIBRI por Lisa Cummins y Derek Bridge de la Universidad College Cork en Irlanda. Estos métodos permiten reducir el tamaño de la base de casos eliminando casos redundantes que no mejoran la eficiencia del sistema. Entre estos algoritmos podemos citar: BBNR (Blame-based noise reduction) y CRR (Conservative Redundancy Removal) de Delany y Cunningham (2004), RENN (Repeated Edited Nearest Neigh-

bor) de Tomek (1976), RC (Relative Cover) de McKenna y Smyth (2000), o ICF (Iterative Case Filtering) de Brighton y Mellish (2002).

Estos algoritmos trabajan en sistemas de clasificación donde la solución de los casos es una única etiqueta o clase. De esta forma es posible predecir la clase de una nueva consulta a partir de las etiquetas de los casos más similares. Esta predicción puede tomarse por unanimidad (`UnanimousVotingMethod`), por mayoría (`MajorityVotingMethod`) o sumando la similitud de los casos (`SimilarityWeightedVotingMethod`).

Evaluación. Como ya se comentó en la Sección 3.1.2.4, los métodos de evaluación se implementaron con la nueva arquitectura de jCOLIBRI 2 en mente. Por lo tanto, su incorporación a esta segunda versión fue completamente directa y se cuenta con los métodos `HoldOutEvaluator`, `LeaveOneOutEvaluator`, `NFoldEvaluator` y `SameSplitEvaluator` descritos en dicha sección.

Extensiones. Las extensiones engloban componentes, métodos y medidas de similitud utilizados para implementar sistemas CBR específicos. jCOLIBRI 2 incluye tres extensiones principales que se detallarán en los siguientes capítulos de esta memoria. En el Capítulo 4 se describirán los componentes para la implementación de sistemas de CBR Textual, mientras que el Capítulo 5 se centra en el desarrollo de aplicaciones con conocimiento intensivo (KI-CBR). Este último conjunto de métodos se fundamenta en la librería *OntoBridge* que posibilita el acceso y manipulación de ontologías. A continuación, el Capítulo 6 se encarga de describir el proceso de implementación de sistemas CBR de recomendación.

También hay que destacar la extensión para visualizar la base de casos. Este componente fue desarrollado por Josep Lluis Arcos (IIIA-CSIC) para mostrar gráficamente la distancia entre los casos. Esta distancia se calcula mediante las medidas de similitud definidas por el usuario y su visualización permite encontrar la mejor configuración para dichas medidas. Por ejemplo, la Figura 3.14 es una captura de este componente en funcionamiento donde los casos se dividen en tres clases distintas. La mejor configuración de las funciones de similitud dividirá perfectamente los tres grupos de casos. Un mejor agrupamiento de los casos según su clase implica que que el método k-NN obtendrá resultados óptimos.

Ejemplos. Los ejemplos incluidos en el armazón tienen dos objetivos diferentes. Por un lado sirvieron de tests durante el desarrollo del armazón para comprobar que toda la funcionalidad se mantenía correctamente al incluir progresivamente nuevas características. Una vez terminado el

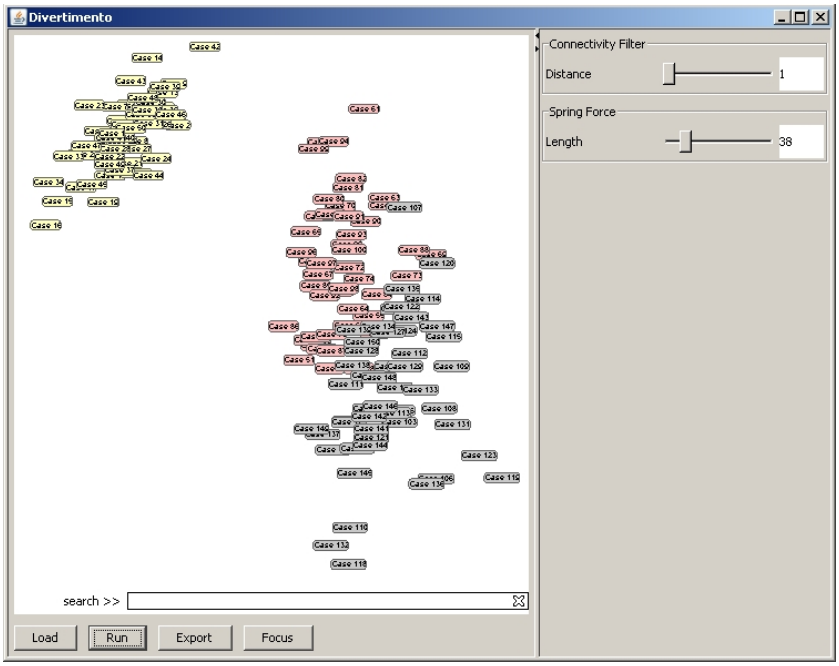


Figura 3.14: Visualización de la base de casos.

desarrollo, estos ejemplos quedaron como parte de la documentación de jCOLIBRI 2 para que lo usuarios puedan acceder al código de diversas aplicaciones finales que utilizan el armazón. Este papel de los ejemplos como mecanismo de documentación se detalla en el siguiente apartado.

3.2.3 Documentación

La documentación de una armazón no es una tarea sencilla y puede realizarse de distintas maneras. En la Sección 2.4.1 se describieron las distintas alternativas para documentar una este tipo de sistemas software: manuales de referencia, tutoriales y aplicaciones de ejemplo. En jCOLIBRI se incluyen estos tres tipos de documentación para facilitar el proceso de aprendizaje del armazón.

Los manuales de referencia incluyen información detallada sobre cada una de las clases del armazón: responsabilidades, métodos, parámetros, etc. Esta documentación se incluye en jCOLIBRI a través de la herramienta JavaDoc[7] que permite generarla automáticamente a través de los comentarios añadidos en el código fuente. Este código fuente también se incluye como documentación junto con diversos diagramas UML de clases y secuencia.

Sin embargo, los manuales de referencia no son demasiado útiles para

[7] http://java.sun.com/j2se/javadoc/

Test	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16
Data Base connector	✓	✓	✓	✓	✓			✓								
Plain Text connector						✓	✓		✓					✓	✓	
Ontology connector										✓						
Custom connectors													✓			✓
KNN	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Enumerated Attributes		✓	✓	✓	✓											
User Defined Types		✓	✓	✓	✓											
Compound Attributes			✓	✓	✓											
Reuse methods			✓	✓	✓											
Revise methods			✓	✓	✓											
Retain methods			✓	✓	✓	✓										
Cases with solution				✓	✓	✓	✓		✓	✓				✓	✓	✓
Attributes mapped to an ontology					✓					✓						
Ontological similarity functions					✓					✓	✓					
Evaluation of CBR systems								✓						✓	✓	✓
Visualization of case bases									✓							✓
Textual CBR methods												✓	✓			✓
Textual similarity methods													✓			✓
Lucene similarity method													✓			
Accuracy Evaluation														✓		
Maintenance Algorithms							✓								✓	
Maintenance Evaluation															✓	
Classification															✓	✓
Test	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16

Figura 3.15: Indice de los ejemplos de jCOLIBRI 2

aprender a usar un armazón ya que se necesita mucho tiempo de estudio del manual antes de poder desarrollar una aplicación, pues no guían al usuario sino que son meramente descriptivos. Para solventar esta carencia también se incluye un completo tutorial (Recio-García et al., 2007b) donde se explica el proceso de desarrollo mediante jCOLIBRI a través de un ejemplo guiado. Este ejemplo guiado consiste en el desarrollo de un recomendador de viajes que permite conocer progresivamente las características del armazón. Además, el tutorial profundiza en el desarrollo de aplicaciones CBR utilizando los distintos elementos de jCOLIBRI: conectores, uso de ontologías, métodos de recomendación, etc.

Como se comentó en la exposición de los distintos sistemas de documentación de armazones, el problema de los tutoriales es que suelen ser demasiado escuetos, quedándose en la superficie del desarrollo de aplicaciones y no profundizan en las funcionalidades más complejas que pueda incluir un armazón. Para resolver esta carencia, las *aplicaciones de ejemplo* suelen consistir en el código fuente de las aplicaciones que los desarrolladores del armazón han construido durante la creación del mismo. Este tipo de documentación es muy económica ya que no se crea ex profeso para documentar sino que las aplicaciones de ejemplo son implementadas durante el desarrollo del arma-

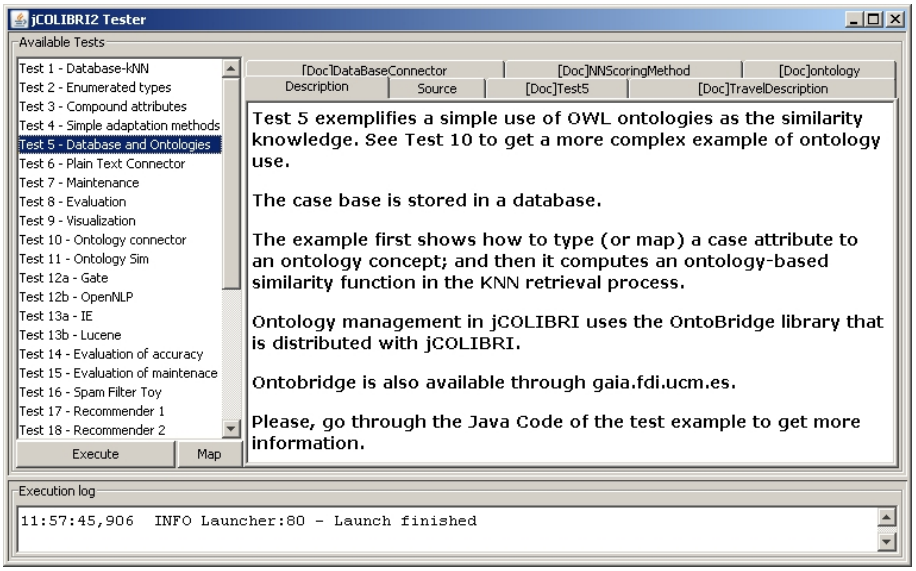


Figura 3.16: Herramienta de acceso a los ejemplos

zón para probarlo y depurarlo. Esta aproximación también se ha incluido en jCOLIBRI y es una de las fuentes de documentación más útiles del armazón. En el apartado anterior ya se comentó que estos ejemplos sirvieron para comprobar el correcto funcionamiento de todas las clases durante el proceso de desarrollo. Una vez concluido, los ejemplos han quedado como documentación del armazón y permiten profundizar en sus distintas características. Para localizar fácilmente los ejemplos que muestran la utilización de cada una de las características del armazón se generó un índice que permite acceder fácilmente a cada uno de ellos y que se organiza según la funcionalidad utilizada. Este índice se muestra en la Figura 3.15, donde se puede comprobar que existen 16 ejemplos sobre sistemas CBR genéricos. Además existe otro índice para los ejemplos de sistemas de recomendación que se detallará en el Capítulo 6.

La documentación mediante ejemplos en jCOLIBRI ha ido un paso más lejos al permitir no sólo el acceso al código fuente, sino permitiendo ejecutar cada uno de ellos a través de una aplicación que también mezcla elementos de otras fuente de documentación. Esta aplicación, mostrada en la Figura 3.16, permite comprobar el funcionamiento real del ejemplo, incluye una breve descripción del mismo, el acceso a su código fuente y el manual de referencia de las clases más importantes en dicho ejemplo. Esta forma de documentar el armazón permitiendo comprobar el funcionamiento de aplicaciones reales generadas con el mismo ha demostrado ser una aproximación muy útil y apreciada por los usuarios según se comprobó en las encuestas presentadas al final de esta memoria.

3.3 Conclusiones

Este capítulo describe las dos versiones de la plataforma jCOLIBRI presentando las características de cada una de ellas y las razones que derivaron en el diseño final de jCOLIBRI 2.

La arquitectura, implementación, evolución y herramientas de jCOLIBRI se encuentran publicados en Recio-García et al. (2005c); González-Calero et al. (2006); Recio-García et al. (2006b); Díaz-Agudo et al. (2007a); Recio-García et al. (2007b)

Inicialmente se ofrece una visión genérica de los conceptos de diseño y arquitectura de jCOLIBRI 1 para después detallar su funcionalidad y características.

En la arquitectura se reutilizan las ideas de UPML para la representación del comportamiento de la aplicación mediante una serie de tareas. Las tares pueden ser sucesivamente descompuestas en tareas más simples hasta que deben ser resueltas por métodos. Dichos métodos son independientes del dominio y reciben la información sobre éste a través de los parámetros durante el proceso de instanciación.

Respecto al diseño de jCOLIBRI 1 hay que destacar su arquitectura en capas. La capa de persistencia representa cualquier medio de almacenamiento de casos: bases de datos, ficheros de texto u ontologías. El acceso a estos medios se realiza a través de objetos denominados conectores que ofrecen una total transparencia a la parte lógica de la aplicación. Cada conector se encarga de leer y escribir los casos de forma adecuada al tipo de persistencia, liberando de dicha tarea a la parte central del armazón.

Los casos son organizados en memoria una vez han sido devueltos por el conector. Mediante las distintas estructuras implementadas (listas, árboles, etc.) se facilita su manejo por parte de los distintos métodos CBR. Al igual que los conectores, los objetos implicados en la organización de la base de casos implementan una interfaz única. De este modo, se ofrece nuevamente una transparencia total al resto de componentes del armazón.

Las aplicaciones en jCOLIBRI 1 se dividen en preciclo, ciclo y postciclo. El preciclo carga la base de datos y recursos necesarios. El ciclo recibe una consulta y ejecuta el ciclo CBR que está dividido en cuatro grandes tareas: recuperación, reutilización, revisión y retención de los casos. Por último, el postciclo se encarga de liberar los recursos.

Cada una de estas tareas puede descomponerse en subtareas, repitiendo el proceso sucesivamente hasta alcanzar el grado de refinamiento requerido. En este punto, el usuario configura los métodos que resuelven las tareas simples por medio de la interfaz gráfica ofrecida por jCOLIBRI 1. Esta interfaz gráfica también permite la definición de la estructura del caso, funciones de similitud, tipos de datos y otro tipo de elementos implicados en el diseño

de los sistemas CBR. Además, se incluyen herramientas de evaluación y de generación dinámica de aplicaciones independientes a partir del diseño de tareas y métodos.

Las distintas herramientas gráficas de jCOLIBRI 1 lo convierten en un buen armazón de caja negra. Sin embargo, esta arquitectura de caja negra no está bien fundada en un diseño de caja blanca claro y extensible. Esto se debe al rápido crecimiento del armazón y algunas decisiones de diseño iniciales donde no se preveía la gran expansión de jCOLIBRI. El acoplamiento entre funcionalidad e interfaz gráfica del diseño de jCOLIBRI 1 dificulta su utilización por los desarrolladores que necesitan controlar la aplicación a nivel de código. Para solucionarlo se desarrolla una nueva arquitectura en jCOLIBRI 2 que soluciona dichos errores aprendiendo de los inconvenientes descubiertos durante más de dos años de desarrollo (Recio-García et al., 2006b).

jCOLIBRI 2 es la segunda versión de la plataforma donde se separan claramente las partes del armazón orientadas a usuarios desarrolladores y a usuarios diseñadores. De esta forma se solucionan los problemas de acoplamiento aparecidos en la versión anterior. En este capítulo sólo se presenta la capa orientada a los desarrolladores, dejando para el Capítulo 6 la descripción de la otra capa ya que en ella se utilizarán las tecnologías de los Servicios Web Semánticos. Esta división en capas de la segunda versión permite implementar un buen armazón de caja blanca que soporte las herramientas de composición pero sin mezclar la funcionalidad.

La parte de jCOLIBRI 2 presentada en este capítulo se centra en su diseño como armazón de caja blanca. En este diseño se mantienen varias características de su antecesora como la arquitectura en capas, la división de las aplicaciones en preciclo, ciclo y postciclo (aunque se hacen algunos añadidos) y la estructuración de los casos. Sin embargo, la implementación de todo este diseño se realiza mediante tecnologías más novedosas y potentes pertenecientes a la plataforma J2EE.

La nueva representación de los casos se aprovecha de estas tecnologías y utiliza Java Beans en lugar del patrón Composite aplicado en jCOLIBRI 1. Los JBeans facilitan enormemente el manejo de los casos por parte de los métodos, su persistencia automática gracias a la librería Hibernate, y su inclusión en aplicaciones web.

Esta última característica de la compatibilidad web es muy importante en jCOLIBRI 2 y por lo tanto se permite utilizar el armazón como una librería de métodos que permita su integración en este tipo de aplicaciones.

jCOLIBRI 2 añade nuevos componentes, métodos y medidas de similitud a las ya existentes en la versión anterior. Se ofrecen tres conectores para bases de datos, textos y ontologías. También se implementan varios métodos de

recuperación y adaptación, junto a otros métodos organizados mediante extensiones que amplían la funcionalidad del armazón y que suponen una gran aportación al desarrollo de aplicaciones CBR. Estas extensiones se describirán en los próximos capítulos de esta memoria.

Capítulo 4

CBR Textual en jCOLIBRI

– ¡Ya tengo lista la caja de herramientas!
Ahora empezaré a fabricar clavos.

Existen numerosos dominios donde los casos manejados por las aplicaciones CBR están representados como textos: informes empresariales o científicos, textos legales, informes de incidentes, etc. Por lo tanto, dentro del campo del CBR existen una serie de técnicas encargadas de manipular este tipo de casos que se engloban dentro de una subárea del CBR denominada *CBR Textual* (comúnmente conocido como TCBR por sus siglas en inglés provenientes de Textual CBR).

Los orígenes, evolución y características de este campo ya se describieron en la introducción de la Sección 2.1.4. Como ese apartado ponía de manifiesto, los sistemas TCBR existentes han sido desarrollados para dominios específicos: búsqueda en FAQs (Burke et al., 1997a), textos legales (Brüninghaus y Ashley, 1999), sistemas de preguntas (Lenz y Burkhard, 1997), ... Por ello, es difícil diseñar un armazón que pueda cubrir las necesidades de la comunidad de CBR Textual. Sin embargo, en jCOLIBRI se presenta una aproximación independiente del dominio que ofrece a los desarrolladores la funcionalidad suficiente para crear este tipo de sistemas. A grandes rasgos, la extensión de CBR Textual incluida en el armazón contiene una serie de métodos que permiten el procesamiento de los textos para estructurar su información, ciertas medidas de similitud específicas, y otros componentes auxiliares.

La heterogeneidad de este tipo de aplicaciones provoca que no haya ningún estándar o consenso sobre la estructura de un sistema de CBR Textual. Esto se debe principalmente a los diferentes requisitos de cada dominio. Por ejemplo, en aplicaciones de clasificación normalmente se necesita un procesamiento simple del texto basado principalmente en un tratamiento sintáctico de los textos. En este escenario, suelen utilizarse únicamente un extractor de raíces y alguna función de similitud basada en el coseno del espacio de

vectores Sin embargo, en otro tipo de aplicaciones se aplican métodos más avanzados de Procesamiento del Lenguaje Natural (NLP, del inglés Natural Language Processing). Podemos encontrar ejemplos de este último tipo de aplicaciones en Brüninghaus y Ashley (2001) y Brown et al. (1998).

Esta diversidad en las aplicaciones TCBR provoca que en jCOLIBRI distingamos dos grandes grupos de técnicas:

CBR Textual Semántico. Los métodos de TCBR semántico extraen información del texto y la representan en los atributos de los casos. Una vez que los casos han sido "rellenados" con esta información, es posible aplicar las técnicas clásicas de CBR que trabajan con casos estructurados. Este tipo de técnicas se califican como semánticas porque intenta "entender" el significado de los textos y organizar su contenido de forma estructurada para poder razonar con él. La gran mayoría de técnicas y sistemas de CBR Textual existentes pertenecen a este grupo.

CBR Textual Estadístico. La recuperación es la etapa más importante en el Razonamiento Basado en Casos y esta característica se acentúa aún más en el subárea del CBR Textual. Además, como el proceso de adaptación de textos es extremadamente complejo, existen numerosos sistemas que se fundamentan principalmente en el proceso de recuperación de los textos. En este escenario es donde aparecen los métodos estadísticos aplicados al CBR Textual que tienen su origen en otras áreas de investigación como la de Recuperación de Información (IR del inglés Information Retrieval). Estas áreas han desarrollado métodos muy eficaces para la recuperación de textos debido en gran parte a la creciente importancia de las búsquedas de documentos web. La principal diferencia de este tipo de métodos con respecto al grupo de métodos semánticos es que no tienen en cuenta la información representada en el documento para computar la similitud. En su lugar utilizan únicamente medidas estadísticas basadas en la frecuencia de aparición de los términos.

Aunque las técnicas de CBR semántico sean las más extendidas existen ciertas situaciones donde su coste de aplicación supera el beneficio de rendimiento obtenido. Los métodos semánticos requieren una gran cantidad de información sobre el dominio y su implantación dentro de las aplicaciones CBR es mucho más compleja que en el caso de los métodos estadísticos. Por lo tanto, si una aplicación va a realizar poca adaptación o no se dispone de conocimiento del dominio, los métodos estadísticos pueden ser una posibilidad muy viable a la hora de implementar un sistema de CBR Textual. En Recio-García et al. (2007c) se presentan una serie de resultados que justifican esta argumentación.

La extensión textual de jCOLIBRI ha ido evolucionando con las distintas versiones del mismo. En la primera sólo se incluían métodos para implementar sistemas de TCBR semántico, y es en jCOLIBRI 2 donde se completan estos métodos y se añaden otros tantos pertenecientes al grupo de TCBR estadístico.

Como la gran mayoría de sistemas pertenecen al grupo de TCBR semántico y esta es la solución más ampliamente adoptada, nos referiremos a este grupo simplemente como aplicaciones de TCBR. Cuando nos refiramos a los métodos del otro grupo, indicaremos específicamente que se trata de TCBR estadístico.

4.1 CBR Textual Semántico

En la introducción al CBR presentada en el Capítulo 2 se comentaba que la gran mayoría de algoritmos y técnicas de CBR requieren que los casos sobre los que se aplican estén representados de forma estructurada. Sin embargo, también se resaltaba que en la gran mayoría de dominios donde el CBR es útil, los casos suelen encontrarse en forma de textos en lenguaje natural: informes, textos legales, etc. Por ello, el CBR Textual (semántico) engloba las técnicas que permiten convertir dichos textos en casos estructurados o, al menos, semiestructurados. De esta forma, se posibilita que las aplicaciones CBR puedan comparar, recuperar y adaptar los casos mediante los métodos CBR "estándar" que trabajan sobre este tipo de representación.

El cometido principal de estos métodos es analizar los textos de un dominio y construir casos semiestructurados donde se represente la misma información que la expresada a través del lenguaje natural. Esta tarea no es sencilla y suele requerir una importante cantidad de información sobre el dominio en cuestión. Por lo tanto, la implementación de un armazón de CBR Textual donde se incluyan métodos independientes del dominio requiere de un buen diseño que orqueste todos los componentes involucrados.

En esta sección se expondrán las características del diseño utilizado en jCOLIBRI para implementar la extensión de CBR Textual y el modelo teórico sobre el que se fundamenta. A continuación se introducirán las distintas tecnologías de NLP utilizadas y cómo se realiza la estructuración de los textos para permitir su procesamiento. Finalmente se detallarán los métodos y medidas de similitud incorporados en el armazón.

4.1.1 Características de un armazón genérico de TCBR

El reto principal en el diseño de un sistema genérico de TCBR es definir la estructura que permita representar los textos como casos. Muchos sistemas CBR trabajan con casos que contienen el texto plano directamente, sin embargo, normalmente se necesitan representaciones que combinen textos y

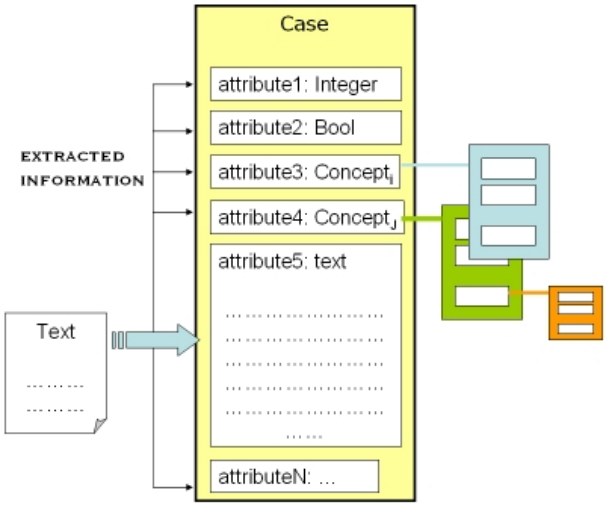


Figura 4.1: Representación de casos para CBR textual

datos estructurados. Esta misma conclusión se expone en Wilson y Bradshaw (1999) donde los autores describen un ejemplo sobre una aplicación de asistencia técnica. En este sistema el operador asigna valores a algunos atributos del producto y además escribe anotaciones textuales en otros. Por lo tanto, se utiliza una representación mixta que es necesaria para realizar el proceso de reutilización de forma eficiente.

Aunque la forma de representar los casos en jCOLIBRI varíe entre ambas versiones, el esquema genérico basado en atributos simples y compuestos se mantiene invariable. Esto permite incorporar fácilmente las estructuras necesarias para representar los casos textuales. Para ello, simplemente se añaden una serie de objetos para implementar los atributos que necesiten almacenar y organizar los textos. Este nuevo tipo de atributo se denomina atributo textual del caso y puede ser utilizado en combinación con otros atributos simples. Esta representación es muy versátil porque permite extraer información del texto y almacenarla los distintos atributos del caso, pero además mantener ese mismo texto dentro del caso por medio de un atributo textual. La figura 4.1 muestra esta aproximación.

Una vez que la estructura del caso está definida, la siguiente decisión consiste en identificar la funcionalidad que debe ser ofrecida en un armazón de CBR Textual. Normalmente los métodos de procesamiento del lenguaje necesitan bastante conocimiento del domino y por lo tanto es muy importante separar este elemento al diseñar los métodos de un armazón. Además dichos métodos deberán ser compatibles con los métodos de CBR genérico preexistentes ya que se utilizarán de forma conjunta. Esto se debe a que en la mayoría de las aplicaciones primero se utilizan métodos de CBR Tex-

tual para procesar los textos y extraer la información, y luego se aplican los métodos CBR genéricos encargados del razonamiento con los casos.

Aunque es difícil establecer una funcionalidad común entre distintos sistemas TCBR, algunos diseñadores han intentado definir diferentes requisitos para este tipo de sistemas (Gupta y Aha (2004), Weber et al. (2001)). Para el diseño de la extensión textual en jCOLIBRI, nosotros hemos elegido el modelo en capas de Lenz (1998). Este modelo se toma como arquitectura de referencia ya que es el diseño más genérico y su flexibilidad permite diferentes combinaciones y extensiones de los diferentes mecanismos de procesamiento de textos.

Aparte del modelo teórico –que se explicará en el siguiente apartado– desde nuestro punto de vista cualquier armazón de CBR textual debería ser diseñado y desarrollado teniendo en mente dos perspectivas distintas:

- Las tecnologías involucradas. Ya que la utilización de textos como fuente de información implica tanto Procesamiento del Lenguaje Natural como algoritmos de Recuperación de Información.

- Relación con el dominio. Estas tecnologías pueden utilizar información dependiente o independiente del dominio. Por lo tanto hay que diseñar un armazón de TCBR que incluya funcionalidad independiente del dominio, pero también mecanismos genéricos y extensibles que puedan ser aplicados con conocimiento del dominio.

Estas dos perspectivas quedan reflejadas en las distintas capas del modelo descrito en la próxima sección y en el que se basa la descomposición a nivel de conocimiento del procesamiento de casos textuales en jCOLIBRI.

4.1.2 El modelo teórico de Lenz para TCBR

Como ya se indicó anteriormente, el modelo elegido para la extensión de TCBR en jCOLIBRI es el propuesto por Lenz (1998). Este modelo se basa en diferentes capas que van procesando los textos gradualmente:

Palabras clave (Keyword layer). Esta capa divide los textos en términos, descarta las palabras vacías (denominadas comúnmente como stopwords: preposiciones, conjunciones,...), y calcula estadísticas sobre la frecuencia de los términos. Esta capa también incluye un análisis gramatical que será utilizado por las capas posteriores. Este paso es independiente del dominio, por lo que puede ser compartido entre distintas aplicaciones CBR.

Expresiones (Phrase layer). Identifica expresiones específicas del dominio mediante un diccionario. La principal dificultad de esta etapa es que

los términos que forman la expresión pueden aparecer de forma no consecutiva en el texto y que las reglas que obtienen las expresiones deben ser generadas manualmente.

Tesauro. Esta capa identifica sinónimos y términos relacionados. Los métodos implementados en esta etapa se utilizan para relacionar los términos de la consulta con los casos. Esta fase es independiente del dominio y para textos en inglés se suele utilizar WordNet (Miller et al., 1990) como tesauro.

Glosario. Esta capa es análoga al tesauro aunque dependiente del dominio concreto de la aplicación. Por lo tanto, es conveniente definir una interfaz común para ambas capas. El principal inconveniente de este paso consiste en la adquisición del glosario.

Características (Feature Value). Esta capa se utiliza en aplicaciones que utilizan casos semiestructurados. Se encarga de extraer características del caso y representarlas en forma de pares <atributo,valor>. Este paso es también específico del dominio.

Clasificación en el dominio (Domain Structure). Utiliza las capas anteriores para clasificar los documentos a alto nivel. Esta etapa asigna etiquetas a los casos que pueden ser útiles a la hora de indexarlos y recuperarlos.

Extracción de información. Esta capa se encarga de extraer partes de los textos (o información sobre los mismos) y representarla de forma estructurada. Esta etapa puede solapar con las dos anteriores.

En la mayoría de estas capas se necesita aplicar mecanismos de Procesamiento del Lenguaje Natural que realicen sucesivas transformaciones en los los textos hasta prepararlos para extraer y organizar la información. Este tipo de algoritmos suelen ser muy comunes en los campos de Recuperación de Información y Procesamiento de Lenguaje Natural. Por lo tanto existen algunas librerías que contienen implementaciones de los mismos y que pueden ser reutilizadas para desarrollar los métodos del modelo de Lenz.

Al implementar la extensión textual de jCOLIBRI se realizó una comparación de las capacidades de las librerías existentes y se seleccionaron dos implementaciones diferentes: OpenNLP (OpenNLP, 2008) y GATE (GATE, 2008). La primera de ella se utilizó en la extensión textual inicial de jCOLIBRI 1, mientras que la segunda se añadió en otra implementación de los métodos de TCBR que se realizó en jCOLIBRI 2. A continuación pasaremos a detallar los fundamentos y características de ambas librerías para después explicar cómo se utilizaron al implementar los distintos métodos.

4.1.3 NLP mediante Máxima Entropía

Los algoritmos de NLP utilizados en los métodos textuales de jCOLIBRI 1 se basan en una técnica denominada *Máxima Entropía*. Esta técnica es muy eficiente y extensible. Podríamos definir la Máxima Entropía como "modelar todo lo que es conocido y no asumir nada sobre lo que se desconoce". En otras palabras, dada una colección de hechos, se elige un modelo consistente con todos esos hechos pero, a la vez, todo lo uniforme que sea posible (ver Manning y Schütze (1999) y Nigam (1999) para más detalles).

Desde el punto de vista de la ingeniería, utilizar Máxima Entropía es una forma excelente de crear aplicaciones que realizan eficientemente tareas de clasificación muy difíciles. Esta afirmación se demuestra en el trabajo de Ratnaparkhi (1998), donde el autor argumenta que esta técnica permite ejecutar Procesamiento del Lenguaje Natural con resultados similares a los de otras tecnologías punteras en ese área. La principal razón es que en los problemas de NLP es imposible encontrar un conjunto completo de ejemplos de entrenamiento. Esto es debido a la variabilidad del lenguaje, y por lo tanto no es correcto extraer restricciones desde los ejemplos. Como la técnica de Máxima Entropía no realiza asunciones al generar el modelo resulta ser una técnica muy apropiada.

Para ilustrar la idea, consideremos el siguiente ejemplo: supongamos la tarea de estimar una distribución de probabilidad p definida sobre $x, y * 0, 1$. Los únicos hechos conocidos sobre p son que $p(x, 0) + p(y, 0) = 0{,}6$ y $p(x, 0) + p(y, 0) + p(x, 1) + p(y, 1) = 1$. El problema consiste en encontrar los valores marcados con "?" en la Tabla 4.1a. Existen muchas formas consistentes de rellenar esas celdas; por ejemplo, la mostrada en la Tabla 4.1b. El principio de la Máxima Entropía recomienda los valores mostrados en la Tabla 4.1c, que es la asignación de probabilidades más genérica que cumple las restricciones de p.

El modelo de Máxima Entropía ha sido implementado y aplicado a tareas Lenguaje Natural en el paquete *OpenNLP*. Este paquete se divide en métodos independientes que pueden ser utilizados de forma aislada. Se incluyen métodos para la eliminación de palabras vacías, identificación de sentencias, análisis léxico, análisis sintáctico, etc. La principal ventaja de esta implementación es que puede ser extendida fácilmente ya que es posible entrenar sus algoritmos con nuevos datos (textos) para adaptarlos a nuevos dominios y aplicaciones. Por ejemplo, podemos adaptar el algoritmo para trabajar en otros idiomas distintos del inglés si contamos un un conjunto de textos de entrenamiento en cualquier otro idioma.

p(a,b)	0	1	
x	?	?	
y	?	?	
total	.6		1.0

(a) Distribución desconocida

p(a,b)	0	1	
x	.5	.1	
y	.1	.1	
total	.6		1.0

(b) Una posible distribución

p(a,b)	0	1	
x	.3	.2	
y	.3	.2	
total	.6		1.0

(c) Distribución Máxima Entropía

Tabla 4.1: Ejemplo de Máxima Entropía

4.1.4 NLP mediante GATE

GATE (GATE, 2008) es una infraestructura para el desarrollo de componentes software que procesan lenguaje natural. Esta librería es ampliamente utilizada en aplicaciones de Extracción de Información e incluye varios componentes que se utilizan en la extensión de TCBR en jCOLIBRI: analizador gramatical, extractor de raíces, extractor de expresiones y extractor de características.

La implementación de los métodos de CBR Textual mediante la librería GATE se incluyó en jCOLIBRI 2 para ofrecer al desarrollador distintas posibilidades al diseñar las aplicaciones. Este proceso fue bastante natural y directo porque nuestro armazón ya incluía una representación de casos e interfaces para los métodos que implementan el modelo de Lenz mediante OpenNLP. Por lo tanto sólo hubo que implementar dichas interfaces mediante los métodos ofrecidos por la nueva librería.

Respecto a las características de la librería, en GATE destacan dos elementos principales: los diccionarios y el reconocedor de expresiones regulares JAPE.

Gate utiliza diccionarios (denominados Gazetteers) que consisten en listas de nombres (ciudades, organizaciones, días de la semana, etc.). Estas listas se utilizan normalmente para asistir en tareas de reconocimiento de entidades, aunque pueden ser utilizadas para cualquier propósito.

Sin embargo, el elemento más importante de GATE es JAPE (Java Annotation Patterns Engine). JAPE permite reconocer expresiones regulares en elementos o anotaciones de los elementos. De este modo podemos reconocer expresiones que aparecen en el texto para anotarlo, pero a su vez podemos aplicar las reglas de JAPE sobre anotaciones previas para obtener anotaciones más complejas. De este modo, GATE incluye métodos que permiten implementar gran parte de las capas del modelo de Lenz: análisis gramatical, características, expresiones, etc. Como las reglas pueden ser definidas por el desarrollador, GATE nos permite implementar métodos que resuelven también las etapas dependientes del dominio.

Desde el punto de vista de la arquitectura de GATE, los métodos men-

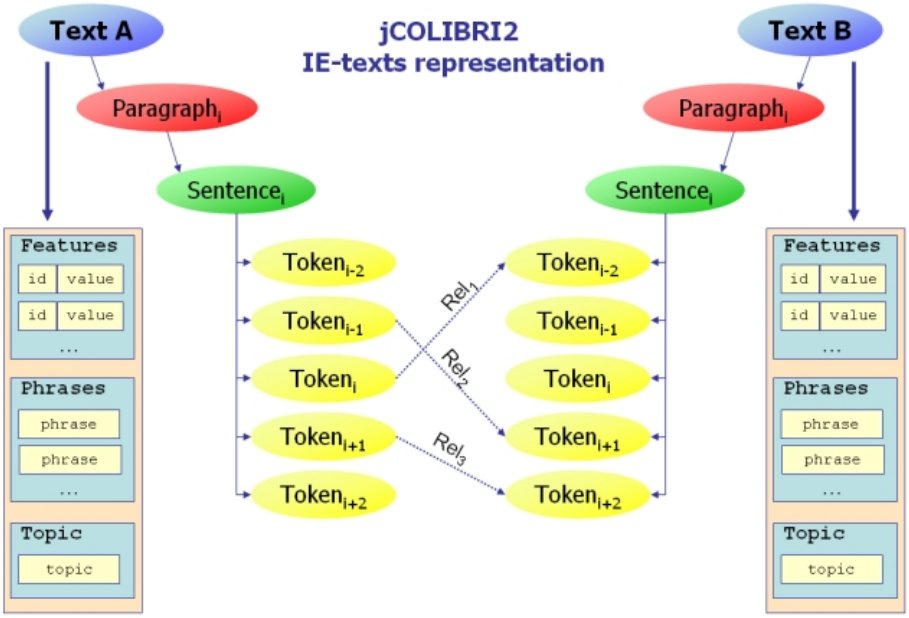


Figura 4.2: Representación completa de los textos

cionados anteriormente se se engloban en el sistema ANNIE (A Nearly-New IE system). ANNIE utiliza máquinas de estados finitos más el sistema JAPE para extraer información de los textos (Cunningham et al., 2002). En este sistema se incluyen las estructuras que sirven de base común a los distintos métodos de la librería.

4.1.5 Representación de textos en jCOLIBRI

La extensión textual de jCOLIBRI define un tipo de datos específico para representar los textos. Mediante este tipo se definen los atributos del caso que contendrán el texto que se utiliza como fuente de experiencia dentro del ciclo CBR. Existe un objeto genérico denominado `Text` que puede ser utilizado tanto por métodos de TCBR semántico como estadístico. En el caso particular de los métodos de CBR semántico se utiliza una especialización denominada `IEText` que permite organizar los textos en párrafos, sentencias y elementos, según se muestra en la Figura 4.2. A continuación describiremos los distintos elementos que aparecen en este esquema y su papel en la representación de la información sobre el texto

Los elementos (Tokens) representan palabras en el texto. Estos objetos almacenan la siguiente información:

- Si el elemento es una palabra vacía.
- Si el elemento es un nombre principal de la frase (núcleo del sujeto u

objeto directo).

- La raíz de la palabra.
- La etiqueta léxica (determinante, nombre, verbo, adjetivo,...).
- Una lista de relaciones con otros elementos similares.

La organización en párrafos, sentencias y elementos es específica de la implementación elegida ya que puede realizarse mediante distintos algoritmos. Como GATE y OpenNLP utilizan métodos distintos pueden obtener resultados diferentes incluso trabajando sobre el mismo conjunto de textos.

La información extraída del texto mediante los distintos métodos del armazón se almacena directamente en los objetos `IEText`. El tipo de información almacenada dependerá de los métodos que hayan sido aplicados al texto:

- Expresiones identificadas en el texto (Phrases).
- Características extraídas en formato <atributo,valor> (Features).
- Identificadores de alto nivel (Topics).

Por ejemplo, en la aplicación sobre restaurantes presentada al final del capítulo se utilizan expresiones para detectar el tipo de comida, las características se utilizan para obtener los distintos platos y los identificadores de alto nivel para clasificar los restaurantes. Toda esta información no está asociada a ninguna frase o párrafo particular por lo que se almacena en el objeto `IEText`.

El modelo de Lenz también define algunos métodos que relacionan elementos de la consulta con elementos del caso. Este tipo de métodos suele buscar relaciones entre los términos mediante un glosario o tesauro. Las relaciones generadas contienen además un valor que identifica la similitud entre los elementos y que es utilizado por los métodos de similitud que trabajan sobre los textos.

4.1.6 Métodos TCBR implementados

Como hemos visto en las secciones anteriores, la extensión de CBR Textual de jCOLIBRI incluye varios métodos que implementan las distintas capas del modelo de Lenz. Dependiendo de la implementación a utilizar (GATE u OpenNLP) se utilizarán diferentes especializaciones de la clase que representa los textos `IEText`. Por lo tanto sólo se podrán combinar métodos que utilicen la misma representación. También hay métodos genéricos que pueden ser aplicados tanto a la representación de GATE como a la de OpenNLP.

La Tabla 4.2 muestra los diferentes métodos de la extensión que son detallados a continuación:

Implementación	OpenNLP	GATE	Genérica
Representación	IETextOpenNLP	IETextGate	IETextOpenNLP, IETextGate, IEText
Paquete	IE.opennlp	IE.gate	IE.common
Capas			
Estructurar texto	OpennlpSplitter	GateSplitter	
Palabras vacías			StopWordsDetector
Extraer raices			TextStemmer
Análisis léxico	OpennlpPOStagger	GatePOStagger	
Nombre principales	OpennlpMainNamesExtractor		
Expresiones		GatePhrasesExtractor	PhrasesExtractor
Glosario			GlossaryLinker
Tesauro			ThesaurusLinker
Características		GateFeaturesExtractor	FeaturesExtractor
Clasificación			DomainTopicClassifier
IE			BasicInformationExtractor

Tabla 4.2: Métodos de la extensión de CBR Textual

OpennlpSplitter y GateSplitter se encargan de dividir el texto en párrafos, sentencias y elementos. El primer método se implementa mediante un algoritmo de Máxima Entropía de OpenNLP, y el segundo mediante JAPE y una serie de diccionarios (gazeteers) que distinguen los signos de puntuación en las abreviaciones de los de final de frase.

StopWordsDetector. Este método funciona con ambas implementaciones y utiliza una lista para identificar las palabras vacías.

TextStemmer se encarga de extraer las raíces de los textos. Para ello se utiliza el paquete SnowBall (SnowBall, 2008).

OpennlpPOStagger y GatePOStagger realizan el análisis gramatical del texto. La primera implementación utiliza un algoritmo de Máxima Entropía y la segunda expresiones regulares.

OpennlpMainNamesExtractor obtiene los nombres más importantes de una frase. El método es estadístico y se basa en los textos de entrenamiento utilizados.

GatePhrasesExtractor y PhrasesExtractor. El primer método utiliza un gazeteer de GATE para obtener las expresiones. El segundo método es genérico y utiliza un lenguaje de expresiones regulares.

GlossaryLinker. Utiliza WordNet para relacionar elementos de la consulta y los casos. Para realizar este enlace se buscan ambos elementos en WordNet y se calcula la distancia dentro de esta ontología lingüística.

ThesaurusLinker. Enlaza elementos de la consulta y casos mediante un tesauro definido por el desarrollador del aplicación. Este tesauro se especifica en un fichero de configuración.

GateFeaturesExtractor y FeaturesExtractor extraen características de los textos mediante expresiones regulares. La primera implementación

utiliza la sintaxis de JAPE, mientras que la segunda utiliza un lenguaje genérico.

DomainTopicClassifier añade una etiqueta al texto basándose en las expresiones y características extraídas anteriormente. Este método utiliza un fichero de configuración donde se definen las diferentes etiquetas.

BasicInformationExtractor extrae información del texto y la almacena en otros individuos del caso. Este método implementa la idea mostrada en la Figura 4.1. Si el texto contiene una característica etiquetada igual que uno de los atributos del caso, se copia el valor de dicha característica al atributo. Esta aproximación es una implementación básica de la última capa de Lenz. Dependiendo de los requisitos de la aplicación y de la información representada en el caso, el desarrollador puede substituir este método por uno propio que extraiga la información según sea necesario. Obviamente, el desarrollo del código necesario es bastante trivial al apoyarse en toda la información obtenida por los métodos aplicados en las etapas anteriores.

4.1.7 Medidas de similitud entre textos

Una vez extraída la información de los textos y almacenada en los atributos del caso se utilizan métodos de recuperación típicos para obtener los casos más similares a la consulta. Estos métodos comparan los casos y consulta utilizando medidas de similitud sobre cada atributo. De igual manera que se comparan atributos de tipos básicos (números, booleanos, cadenas, ...) es posible calcular la similitud entre los textos. jCOLIBRI incluye diversas medidas clásicas de similitud para realizar la comparación:

Cosine

$$cosine(t_1, t_2) = \frac{|(t_1 \cap t_2)|}{\sqrt{|t_1|, |t_2|}}$$

Dice

$$dice(t_1, t_2) = \frac{2 * |(t_1 \cap t_2)|}{(|t_1| + |t_2|)}$$

Jaccard

$$jaccard(t_1, t_2) = \frac{|(t_1 \cap t_2)|}{(|t_1| \cup |t_2|)}$$

Overlap

$$overlap(t_1, t_2) = \frac{|(t_1 \cap t_2)|}{\text{mín}\,(|t_1|, |t_2|)}$$

Compression

$$CDM(x, y) = \frac{C(xy)}{C(x) + C(y)}$$

donde $C(x)$ es el tamaño del texto x después de comprimirlo (análogamente con $C(y)$) y $C(xy)$ es el tamaño después de comprimir la concatenación de ambos textos.

Este método ha sido desarrollado para jCOLIBRI en colaboración con el University College Cork (Irlanda) . En Keogh et al. (2004); Delany y Bridge (2006) se describe el fundamento lógico de esta medida donde se utiliza la compresión del texto para medir la similitud. Consiste principalmente en que si los textos x e y son similares, al comprimirlos juntos se obtendrá un archivo más pequeño que si se comprimen por separado.

Normalised Compression

$$NCD(x, y) = \frac{C(xy) - \text{mín}\,(C(X), C(y))}{\text{máx}\,(C(x), C(y))}$$

Este método es análogo al anterior aunque normalizando los valores.

4.2 CBR Textual Estadístico

En la sección anterior se han expuesto los detalles de los métodos de TCBR semánticos. Sin embargo, la segunda versión de jCOLIBRI incluye otro conjunto de métodos que pertenecen al grupo de técnicas TCBR estadísticas. Como ya se ha mencionado, estos métodos aprovechan los resultados obtenidos en el área de Recuperación de Información. Aunque nuestra intención inicial era distinta, nuestro trabajo nos ha llevado a formular la hipótesis de que los métodos estadísticos pueden obtener mejores resultados que los métodos semánticos bajo determinadas circunstancias. En este tipo de sistemas CBR, la recuperación no consistiría en extraer información de los textos y almacenarla en otros atributos del caso, sino en comparar directamente el texto de la consulta con los textos de los casos y recuperar los más similares.

El método de recuperación estadístico implementado en jCOLIBRI 2 utiliza el motor de búsqueda *Apache Lucene* (Hatcher y Gospodnetic, 2004).

Lucene es una librería de código abierto muy extendida y probada. Siendo útil en cualquier aplicación que requiere indexación y búsqueda de textos, Lucene ha obtenido un gran éxito por su utilidad en la implementación de motores de búsqueda para Internet o bases de documentos locales. Este motor se basa en el Modelo de Espacio Vectorial (VSM, del inglés Vector Space

Model, (Salton et al., 1975)) y el modelo Booleano para determinar cómo de relevante es un documento respecto a la consulta de usuario. VSM se basa en la idea de contar el número de ocurrencias de los términos de la consulta en los documentos. El modelo Booleano permite ordenar los documentos de acuerdo a condiciones expresadas en la consulta mediante el uso de la lógica booleana (que aparezcan $A \wedge B$, $A \vee B$, $\neg A$, etc). Lucene optimiza estos dos modelos para mejorar las búsquedas pero en general su funcionamiento se basa en estas ideas tan simples.

Mediante la utilización de Lucene, el método de recuperación de casos estadístico que se incluye en jCOLIBRI 2 permite recuperar casos textuales de forma muy eficiente y rápida. Sin embargo, la diferencia fundamental de este método con respecto a los anteriores es que no se puede utilizar una representación estructurada de los casos ya que no se obtiene conocimiento sobre la información semántica de los textos.

A priori la intuición es que esta forma de recuperar textos no parece demasiado apropiada para sistemas CBR. Sin embargo, al revisar el estado del arte en IR encontramos varios resultados que mejoran en ciertas circunstancias la eficiencia de los métodos "semánticos". Esta afirmación, presentada y discutida en Recio-García et al. (2007c), puede ser bastante controvertida ya que la intuición más extendida es que al "entender" los textos se obtienen mejores resultados. Podemos decir que los métodos semánticos sí "entienden" la información de los textos ya que extraen sus características. Por lo tanto, es natural pensar que una aplicación que maneja el significado de los textos debería obtener mejores resultados que una aplicación que no lo haga. Sin embargo, cuando miramos los resultados reales obtenidos en otros campos de investigación podemos encontrar justamente lo contrario.

En la comunidad de Recuperación de Información, los mejores métodos no utilizan ningún tipo de información semántica y se basan simplemente en cálculos estadísticos. De hecho, los llamados "Buscadores Semánticos" no han conseguido superar los buscadores sintácticos actuales (Google, por ejemplo) y presentan grandes problemas de escalabilidad junto con grandes costes de integración.

Estos problemas se originan en el coste de generación de las reglas específicas de cada dominio y la baja precisión cuando son aplicadas a textos en lenguaje natural. Además, hay que tener en cuenta la complejidad de diseño en las aplicaciones que intentan extraer el contenido semántico del texto.

Este tipo de sistemas suelen estar compuestos por varias etapas de procesamiento del texto. Como consecuencia de su encadenación, una pequeña distorsión en las etapas iniciales se amplifica en las posteriores y se convierte en un error importante del resultado final. De este modo puede ocurrir que los recursos invertidos en generar y modelar las reglas del dominio no tengan una repercusión directa en el rendimiento final del sistema.

Actualmente los métodos sintácticos pueden obtener mejores resultados en determinados escenarios donde la extracción de la información es complicada debido a la dificultad del dominio. Sin embargo es lógico y previsible pensar que los sistemas semánticos que entienden el significado de los textos son el camino de investigación más viable y prometedor. Tanto la comunidad de Web Semántica, como la de Lenguaje Natural y Extracción de Información están realizando grandes avances para mejorar los resultados de la recuperación estadística. Véase por ejemplo: Sable et al. (2002); Kosala y Blockeel (2000); Xiao et al. (2004); Voorhees (1993); Castells et al. (2007).

A continuación detallaremos las distintas técnicas de recuperación de información utilizadas para implementar los métodos de CBR estadístico incluidos en jCOLIBRI.

4.2.1 Clustering

Como se ha explicado en la sección anterior, los métodos estadísticos no obtienen ninguna información sobre los textos a excepción de la puramente estadística. Una forma de enriquecer el acceso y organización de los textos sin contar con conocimiento sobre el dominio es la agrupación de los documentos en conjuntos con información común (conocido comúnmente como *clustering*).

Los algoritmos de clustering más comunes como los jerárquicos o "K-means" (Witten y Frank, 2000b) agrupan los documentos pero no devuelven una descripción de los grupos generados. Sin embargo, Lingo (Osinski et al., 2004) es una algoritmo implementado en el armazón *Carrot2* que permite agrupar los resultados de una búsqueda y además devuelve una breve descripción textual de cada grupo. Lingo se basa en el Modelo de Espacio Vectorial (Salton et al., 1975), Indexación por Semántica Latente (Deerwester et al., 1999) y Descomposición de Valores Singulares (Golub y Kahan, 1965) para asegurar que se obtienen descripciones legibles para los humanos de cada grupo de documentos.

El método de clustering de Lingo se aplica al resultado de una búsqueda de documentos. Por lo tanto puede encadenarse con un motor de búsqueda como Lucene para agrupar los textos devueltos tras una recuperación. De esta forma se convierte en un un complemento perfecto para el método de recuperación de IR que incluye nuestro armazón. La funcionalidad de Carrot2 se encapsula en un método de jCOLIBRI que oculta la complejidad del algoritmo y permite manejarlo fácilmente. Este método se combina con el método de recuperación que utiliza Lucene para obtener casos similares a una consulta dada ofreciendo muy buenos resultados.

Un algoritmo de clustering con etiquetas puede ser utilizado por aplicaciones de TCBR para facilitar la recuperación de los casos ayudando a

elegir el documento más similar. Sin embargo, este método también puede ser aplicado para guiar la adaptación de los textos como se expone en Recio-García et al. (2007c). La adaptación de textos es un proceso muy intensivo en conocimiento y raramente planteado en los sistemas de CBR Textual, así que nuestra propuesta –en la línea de la adaptación supervisada– ofrece al usuario una serie de herramientas interactivas para guiarlo en el proceso de adaptación de los casos. De este modo, el usuario se encarga de realizar la adaptación aunque el sistema encuentra el conocimiento necesario para realizar esta tarea buscando piezas de texto similares.

El método de adaptación propuesto se definiría como *reutilización transformacional*, donde una copia del caso recuperado se utiliza como base sobre la que realizar cambios. Entonces, el usuario realiza cambios mediante un editor de texto: borra fragmentos de texto (palabras, sentencias, párrafos o secciones); escribe nuevos fragmentos; o substituye unos fragmentos por otros similares.

Este método asiste al usuario en el proceso de sustitución ya que utiliza los grupos (clusters) para indicar qué fragmentos en otros documentos están relacionados con el texto que el usuario está editando actualmente.

En resumen, los métodos de TCBR estadístico incluidos en jCOLIBRI permiten realizar un simple proceso de adaptación supervisada donde el usuario realiza cambios basándose en fragmentos de texto relacionados. Estos fragmentos relacionados se identifican mediante los métodos de IR y clustering explicados anteriormente. De esta forma, la adaptación es una extensión de la recuperación, donde el sistema obtiene buenos candidatos para realizar las substituciones.

4.3 Procesamiento de consultas en lenguaje natural

Una parte muy importante en el CBR Textual es el procesamiento de la consulta. Aunque existen diversas aproximaciones para interactuar con un sistema CBR como podrían ser los formularios, el mecanismo más cómodo para los usuarios es el lenguaje natural. Aunque el tipo de interacción varíe dependiendo de la aplicación (preguntas-respuestas, descripción de productos, preguntas específicas,etc.), el procesamiento de la consulta es un elemento vital de todas ellas. Por ejemplo podríamos citar los sistemas SMILE (Brüninghaus y Ashley, 2006), FAQFinder (Burke et al., 1997a) o RealDialog (Branting et al., 2004).

Debido a la utilidad de las consultas en lenguaje natural, su procesamiento cobra un papel clave dentro de los sistemas CBR. Por ello, en Díaz-Agudo et al. (2007b) se presenta una aproximación para obtener y estructurar consultas en lenguaje natural. Para ello, se utilizan los métodos de CBR Textual de jCOLIBRI y las capacidades semánticas de las ontologías. Los detalles sobre

este último componente no son necesarios para comprender el funcionamiento del mecanismo de procesamiento de la consulta, por lo que simplemente se presentará la funcionalidad aportada delegando su explicación al Capítulo 5.

El dominio elegido para ilustrar el procesamiento de consultas es el recomendador de viajes que se incluye como ejemplo en jCOLIBRI. Este recomendador simula una agencia de viajes donde cada caso representa un viaje determinado. La base de casos contiene 1024 casos con 9 atributos cada uno: destino, transporte, hotel, duración, etc. La ontología utilizada para enriquecer las consultas está formalizada en OWL e incluye 244 conceptos, 82 propiedades y 306 instancias[1].

Para ilustrar la utilidad de la ontología supongamos el siguiente ejemplo: C_1 es un caso que describe unas vacaciones a la isla española de *Tenerife*. Ofrece unas vacaciones de dos semanas en un hotel de 3 estrellas para cuatro personas. La temporada es Julio y el precio es 543 euros. Ahora imaginemos un cliente de la agencia de viajes que desea ir con su familia a *Gran Canaria*. Sus preferencias son de un destino de relax con clima agradable pero no desea gastar más de 500 euros. Idealmente el usuario podría utilizar algún tipo de formulario para introducir su consulta o una interfaz de lenguaje natural para describir sus preferencias. La entrada al sistema sería la descripción del caso y la salida un conjunto de soluciones adecuadas (viajes adaptados a sus requisitos). El caso C_1 podría ser presentado al usuario incluso cuando el destino es *Tenerife* ya que el sistema infiere a través de la ontología que *Gran Canaria* y *Tenerife* son ambas islas del archipiélago Canario con características similares.

El proceso de *comprensión*, desde la consulta textual a la representación estructurada sigue los pasos de: Extracción de Información, Sinónimos, Razonamiento con Ontologías y Confirmación del usuario.

La *fase de Extracción de Información* utiliza los métodos de CBR Textual incorporados en jCOLIBRI. Más específicamente se utilizan los métodos de la capa de palabras clave para filtrar las palabras vacías, extraer la raíz de las palabras, realizar el análisis léxico y extraer los nombres principales. Finalmente se realiza la extracción de información mediante los distintos métodos que utilizan reglas para reconocer expresiones regulares y que fueron expuestos en la Sección 4.1.6.

La *fase de sinónimos* mejora la información extraída en el paso anterior. Para ello, el sistema identifica los datos obtenidos con respecto a los valores permitidos para cada atributo. Este proceso se realiza de dos formas distintas:

- WordNet. Permite encontrar sinónimos genéricos. Aunque existen di-

[1] Dicha ontología se encuentra disponible en http://gaia.fdi.ucm.es/ontologies.

versas maneras de utilizar WordNet para resolver esta tarea nuestro sistema solamente busca en cada *synset*[2] por motivos de eficiencia aunque el método de la extensión textual utilizado permitiría obtener relaciones más complejas.

- Sinónimos definidos por el usuario. En este paso se utilizan una serie de sinónimos definidos por el usuario que no aparecerán en WordNet al ser dependientes del dominio. Por lo tanto, el desarrollador tendrá que generar un diccionario específico.

La fase de *razonamiento mediante ontologías* permite encontrar información que no se encuentra directamente en el texto. La inclusión de este paso mejora en gran medida los resultados del procesamiento de la consulta. Como ejemplo utilizaremos la siguiente consulta: "Quiero pasar unos 10 o 15 días en la maravillosa isla de Lanzarote este próximo verano. Por supuesto, me gustaría viajar en avión". Una vez procesada la consulta se obtendría la siguiente representación estructurada: *Consulta (Transporte=Avión; Temporada=Junio Julio Agosto; Duración=10 15; Región=Fuerteventura GranCanaria Tenerife)*. El módulo de procesamiento ha utilizado la ontología para cambiar la información de la temporada extraída del texto *verano* por los valores *Junio*, *Julio* y *Agosto*. Lo mismo ocurre con el destino, donde se detecta que el vocabulario de la consulta es diferente del utilizado en la descripción de los casos. Como *Lanzarote* no es uno de los destinos que ofrece la agencia, se cambia la consulta para incluir cualquiera de las *Islas Canarias* de acuerdo con la información en la ontología.

Finalmente, la fase de *confirmación del usuario* consiste en presentar la consulta interpretada por el sistema de forma estructurada. De esta forma el usuario puede validar la información utilizada para recuperar los viajes.

En la Figura 4.3 se muestra un ejemplo de esta interfaz. A partir de la consulta en lenguaje natural se crea una versión estructurada de la consulta que puede ser editada por el usuario. En esta figura se pone de manifiesto la principal ventaja de nuestra aproximación: el usuario no necesita conocer qué valores están permitidos para describir el viaje. Es decir, si se utilizara un formulario, el usuario debería saber que los valores válidos para el atributo *Temporada* son cada uno de los meses y que no es válido introducir algo como *verano*. Por lo tanto, la ontología sirve para enlazar el vocabulario del usuario o de la consulta con el vocabulario utilizado para describir los viajes.

4.3.1 Evaluación

La evaluación de un módulo de procesamiento de lenguaje natural es una tarea difícil porque es necesario introducir distintas consultas en lenguaje

[2] Un sysnset es un nodo de WordNet que contiene términos con significado equivalente (sinónimos).

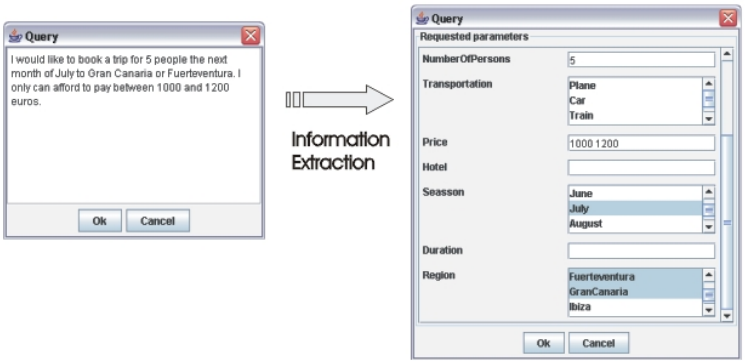


Figura 4.3: Interfaz de consultas en lenguaje natural

natural y los desarrolladores tienen a utilizar consultas *fáciles*. Con este término nos referimos a que el diseñador del sistema tenderá a utilizar el mismo vocabulario que aparece en los casos. Sin embargo se podría decir que el comportamiento es bueno en general ya que al introducir las consultas se obtiene casi toda la información relevante que aparece en el texto.

Para evaluar este módulo se a utilizado la medida de *exhaustividad*[3] respecto al número de términos relevantes de una consulta textual. La idea consiste en medir la relación entre el número de términos extraídos correctamente y el número de términos relevantes que aparecen en la consulta.

La calidad del texto y la adecuación de los atributos buscados son cruciales en el procesamiento del texto. Esto significa que el número de términos relevantes en la consulta depende de las estructura del caso. Supongamos la siguiente consulta:

"Quiero visitar El Cairo el próximo Julio. Quiero estar ahí 10 días visitando la ciudad. El precio debe estar alrededor de los 1000 euros para 2 personas. Nos gustaría alojarnos en un hotel Melia o Hesperia".

En esta consulta aparecen 6 términos relevantes para la consulta: destino, duración, precio, nº de personas, tipo de vacaciones y temporada. Sin embargo en la estructura del caso no se almacena información sobre la compañía a la que pertenece el hotel, así que ese atributo no es tenido en cuenta durante la evaluación.

La exhaustividad mide la proporción de términos que se extraen correctamente por el sistema. En la consulta anterior se obtiene un 100 %. Sin embargo, debido a la estructura del caso, nuestro módulo obtiene un 0 % de exhaustividad para la siguiente consulta:

"Quiero reservar un viaje para mí y mi esposa. No me gusta Mallorca pero cualquier otra isla sería una buena elección. Me da miedo volar así que

[3]Utilizaremos el término "exhaustividad" como traducción del término inglés "recall".

necesito otra alternativa para viajar."

Aquí aparecen 3 términos relevantes: nº de personas, destino y transporte. El problema está en la expresividad del lenguaje utilizado y no en la interfaz de lenguaje natural. Aunque nuestro asustado usuario utilizara un formulario no podría expresar ese requisito ya que normalmente no se ofrece la opción de elegir "Cualquier destino menos ..." o "Cualquier transporte menos...". Este mismo problema ocurre en consultas como: *"Quiero ir a Ibiza o Mallorca"*, *"Quiero gastar menos de 100 euros"* o *"Quiero ir a Ibiza y Mallorca en un viaje combinado"*.

A continuación se muestran una serie de consultas y los resultados obtenidos:

1. *"Me gustaría visitar la maravillosa ciudad de El Cairo el próximo mes de Julio. Quiero permanecer ahí durante 10 días. El precio debe ser más o menos de 1000 euros para 2 personas".*
   - Nº de términos relevantes en la consulta: 5
   - Nº de términos extraídos: 5
   - Exhaustividad: 100 %
   - *Consulta (Destino=El Cairo; Temporada=Julio; Duración=10; precio=1000; Nº personas=2)*
2. *"Quiero un viaje a Fuerteventura de 15 días. Quiero ir con mi novia. El precio debe ser menor de 1000 euros".*
   - Nº de términos relevantes en la consulta: 4
   - Nº de términos extraídos: 3
   - Exhaustividad: 75 %
   - *Consulta (Destino=Fuerteventura; Duración=15; precio<1000)*
3. *"Me gustaría reservar un viaje para pasar las noches del viernes y sábado en un sitio bonito. No me importa el destino pero no quiero volar por avión".*
   - Nº de términos relevantes en la consulta: 2
   - Nº de términos extraídos: 0
   - Exhaustividad: 0 %
   - *Consulta (Transporte=avión)*

En la última consulta, el módulo no es capaz de extraer "transporte=no avión". Pero, como ya se explicó con antelación, este problema se debe a la expresividad del lenguaje de descripción de la consulta.

Para la evaluación del experimento se utilizó un conjunto inicial de 20 consultas *fáciles*. Esto se debe a que se sólo se incluía información sobre atributos que estaban en los casos. A continuación se realizaron las mismas consultas pero a través de un formulario donde se muestra directamente la estructura de representación de los casos. Este proceso nos permite comprobar la eficacia de nuestro módulo de procesamiento de consultas comparando los resultados generados por éste con la consulta introducida en el formulario por el usuario.

En este conjunto de consultas se obtuvo un valor de exhaustividad del 92 %, lo que significa un funcionamiento muy bueno del módulo de procesamiento ya que se extrae casi toda la información de la consulta textual.

El segundo conjunto de consultas fueron formuladas por terceras personas que no conocían la estructura de los casos, ni los valores permitidos para cada atributo. Simplemente se les pidió introducir consultas para una agencia de viajes. El resultado de este conjunto de casos fue del 70 % de exhaustividad. Este valor supone una diferencia muy aceptable con el conjunto anterior y supone un buen resultado. Hay que recordar que en cualquier caso el usuario debe confirmar la consulta y que con este resultado se garantiza que 7 de cada 10 atributos ya son configurados automáticamente por el sistema.

Como conclusión de este apartado, remarcar que el procesamiento de las consultas en lenguaje natural es una característica bastante importante para los sistemas CBR. Este proceso se realiza reutilizando los métodos de TCBR incorporados a jCOLIBRI. Además, se propone un mecanismo de mejora de las consultas mediante el conocimiento representado en ontologías. Para realizar esta etapa se utilizan los métodos que se describirán en el Capítulo 5.

4.4 Resultados Experimentales

En las secciones 4.1 y 4.2 se han descrito los métodos de CBR Textual tanto semántico como estadístico que han sido incorporados en jCOLIBRI. Estos métodos han sido evaluados mediante aplicaciones que permitían cotejar las diversas aproximaciones y en este apartado presentaremos un resumen de los resultados obtenidos.

En Recio-García et al. (2005a) se comparan los métodos de TCBR semántico mediante un sistema recomendador de restaurantes. Dicha comparativa se centra en evaluar la eficiencia de los métodos semánticos de IR+IE (Recuperación y Extracción de Información) frente a los métodos puramente estadísticos de IR. Este experimento se detalla en la Sección 4.4.1. Complementariamente, la Sección 4.4.2 describe el trabajo de Recio-García et al. (2007c) donde se evalúa el proceso de recuperación y adaptación mediante métodos estadísticos.

4.4.1 Recuperación semántica vs. sintáctica

Para evaluar los métodos de la extensión textual de jCOLIBRI desarrollamos un recomendador de restaurantes (Figura 4.4). La base de casos en dicha aplicación contiene 100 descripciones diferentes obtenidas de un sitio web real[4]. Estas páginas contienen textos sobre restaurantes que también fueron utilizados en el proyecto sobre extracción de información Ariadne (Muslea et al., 1998). La información sobre cada restaurante consiste en su nombre, dirección y descripción textual. Por ejemplo:

```
ALEJO'S PRESTO TRATTORIA
4002 Lincoln Blvd., Marina del Rey
(310) 822-0095
While cooking in some of the finest (and most expensive) restaurants
in L.A., chef Alejo had an idea. Why not open a small, intimate
eatery that serves the same gourmet food at lower prices? Alejo's
now has two restaurants serving delectable, homemade Italian
specialties like pasta primavera ($6.95), linguine pescatore
(shrimp, squid and clams, $9.95) and chopped salad (\$4.95). Try
the authentic paella ($10.95) on Friday and Saturday nights, and
don't forget Alejo's famous classic, spaghetti and meatballs
($6.95). Lunch Mon.-Fri., dinner seven nights. Beer and wine
(Westchester); takeout. AE, DIS, MC, V.
```

El objetivo del experimento es comparar el rendimiento de los algoritmos de IR frente a los métodos de IR+IE implementados en jCOLIBRI. Como el experimento se realizó antes de incorporar al armazón el método basado en Lucene, se utilizó la medida coseno (ver Sección 4.1.7) como método de recuperación de IR.

Para poder aplicar Extracción de Información, se identificaron 13 atributos comunes a todas las descripciones de restaurantes: dirección, tipo de comida, precios, horarios, platos, etc. y se desarrolló una pequeña ontología para clasificar la información extraída en 15 tipos de direcciones, 20 tipos de comidas, 13 horarios, y 100 platos. Por último, toda esta información sobre el dominio de la aplicación se completó con diversos conjuntos de reglas que configuran los métodos de CBR textual: las *expresiones* se utilizan para detectar el tipo de comida (carne, pescado, pasta,...), las *características* extraen el nombre de los cocineros, el *glosario* contiene sinónimos sobre el dominio de los restaurantes y la comida, y finalmente la *clasificación* se utiliza para asignar identificadores cada restaurante.

Una vez que el recomendador de restaurantes estuvo implementado y se representó la información sobre el dominio junto con las reglas de extracción de información, los experimentos compararon las siguientes aproximaciones:

- IR con consultas textuales,

[4] http://www.laweekly.com

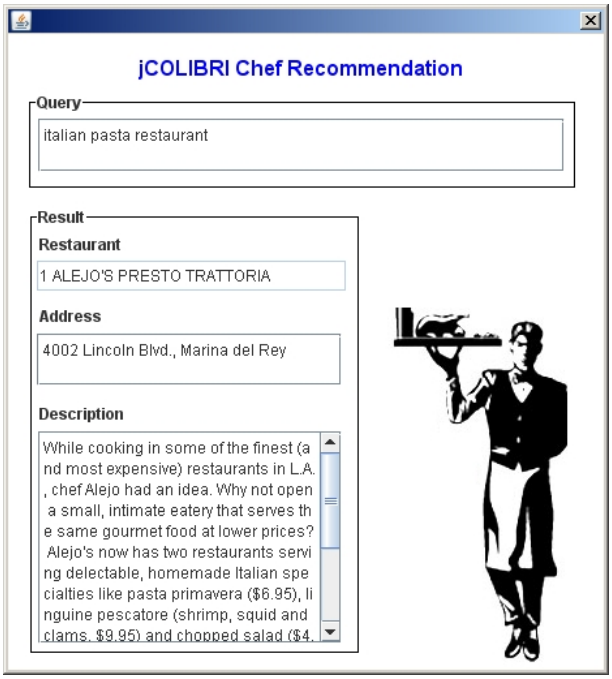


Figura 4.4: Recomendador de restaurantes

- IR + IE con consultas textuales, y
- IR + IE con consultas estructuradas (el usuario elige diversos valores para cada atributo del caso mediante un formulario).

Para el experimento de IR con consultas textuales simplemente se utilizó la similitud coseno entre los casos:$|(o_1 \cap o_2)|/(\sqrt{|o_1|} * \sqrt{|o_2|})$ sin ninguna otra información extraída de los textos.

En el caso de aplicar Extracción de Información con consultas textuales, se compara la similitud entre los atributos del caso y la consulta utilizando la información de la ontología. Para ello se utilizan medidas de similitud locales para la dirección, tipo de comida, horarios y platos. La similitud global se obtiene mediante la siguiente fórmula:

$$\begin{aligned} SimilitudGlobal &= 0{,}2 * Sim(direccion) + 0{,}3 * Sim(tipoComida) + \\ &\quad 0{,}1 * Sim(precio) + 0{,}1 * Sim(horario) + \\ &\quad 0{,}2 * Sim(platos) + 0{,}1 * Sim(otros) \end{aligned}$$

En esta fórmula el horario engloba la similitud entre el horario de desayuno, comida y cena. Y “otros” representa la media de los atributos: “alcohol”, “para llevar”, “a domicilio” y “catering”.

	Location	FoodType	Price	BreakFast	Lunch	Dinner	Dishes	Alcohol	Takeout	Delivery	Catering	Parking
Q	Beverly Hills	Japanese	7	never	never	Sat	beef	true	true	false	true	true
C	Torrance	Japanese	9,95	never	7d	7d	..,beef,...	true	true	false	false	true
LS	0,5	1	0,8525	1	1	1	1	1	1	1	0	1

Tabla 4.3: Ejemplo de consulta estructurada en el recomendador de restaurantes

Por último, la Tabla 4.3 muestra el resultado de comparar una consulta estructurada (fila Q) con un caso al que se han aplicado IR+IE. Este sistema compara la consulta con la representación estructurada de cada caso que se ha obtenido al extraer la información. La fila C de la tabla muestra uno de esos casos. La fila LS contiene los valores de similitud locales obtenidos. Si aplicamos la fórmula de similitud global descrita anteriormente obtendríamos una similitud global de 0,865.

Para comparar cada una de las tres aproximaciones se utilizaron las medidas típicas de *precisión* y *exhaustividad*.

Después de efectuar diversas consultas al sistema se obtuvieron los resultados mostrados en la Figura 4.5. El experimento muestra mejores resultados en la aproximación que utiliza la consulta estructurada. Obviamente, esto se debe a la pérdida de información cuando dicha información estructurada se extrae automáticamente desde el texto. La similitud obtenida al utilizar IR+IE fue casi siempre mayor que 0.75, mientras que con la aproximación de IR pura se obtenían valores alrededor de 0.35. Esa es la razón de la baja *exhaustividad* obtenida en esa aproximación ya que valores tan bajos dificultan la distinción entre los textos realmente relevantes para la consulta.

4.4.2 Recuperación y adaptación estadística

Para evaluar los métodos estadísticos de recuperación y adaptación presentados en la Sección 4.2 se generó una aplicación para el reto (Challenge) del cuarto workshop internacional de TCBR celebrado en 2007. Este reto proponía una serie de textos sobre incidentes aéreos[5]. A partir de ellos debía generarse una aplicación CBR que ayudara al usuario a generar nuevos informes. Por lo tanto debían aplicarse métodos de recuperación y adaptación de TCBR.

Los textos tenían las siguientes características: no contaban con una estructura fija, no se disponía de información sobre el dominio y en la mayoría de casos se utilizaba un lenguaje muy técnico. Por lo tanto, la aplicación de técnicas semánticas de TCBR a estos casos es muy compleja y requiere dedicar numerosos recursos a modelar el dominio y obtener las reglas de IE. Bajo estas premisas, la mejor solución son los métodos estadísticos incluidos

[5] http://www.tsb.gc.ca/en/reports/

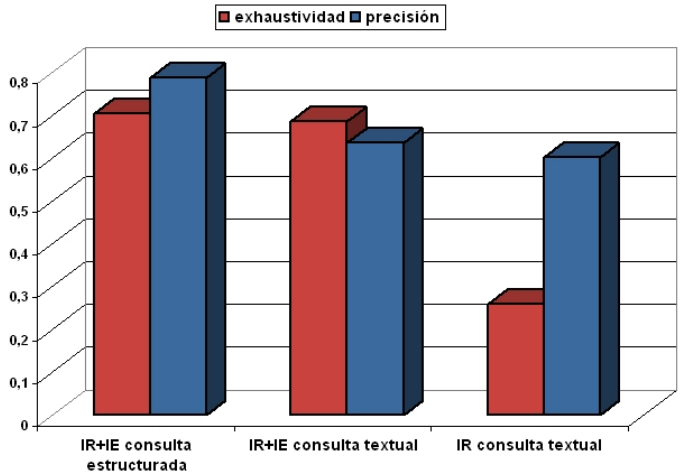


Figura 4.5: Resultados del experimento de restaurantes

en jCOLIBRI.

Nuestra aproximación al problema consiste en encontrar el caso más similar a la consulta mediante la recuperación con Lucene y usar su contenido para guiar al usuario en la generación de un nuevo informe. La estructura del caso recuperado se utiliza para agrupar los otros documentos (mediante Carrot2) y encontrar la información relacionada que ayuda al usuario en la adaptación. Esta aproximación es muy parecida a la presentada en Lamontagne y Lapalme (2004), aunque nuestra aproximación es supervisada y permite manejar textos con diferente estructura.

La aplicación desarrollada para el workshop se denomina *Challenger 1.0*[6] y su funcionamiento se divide en tres etapas principales:

Preciclo o inicialización. Esta parte de la aplicación inicializa WordNet ya que se utilizará para relacionar las distintas secciones de los documentos. Como los textos se dividen en secciones que pueden ir identificadas de forma diferente, se utiliza WordNet para relacionarlas. Por ejemplo las secciones "Summary" y "Synopsis" contienen la misma información y han de ser relacionadas. A continuación un conector jCOLIBRI específico carga los archivos HTML analizándolos y estructurando su contenido de acuerdo a las etiquetas de cabecera `h1,h2,...`. Cada caso contendrá la URL del texto y sus secciones (tanto en formato HTML con en texto plano). Una vez que los textos han sido cargados, se crea el índice inverso de términos-documentos que utiliza Lucene en sus búsquedas. Este índice es creado a nivel de secciones para permitir

[6] Disponible en: http://gaia.fdi.ucm.es/grupo/software.html

Recuperación.

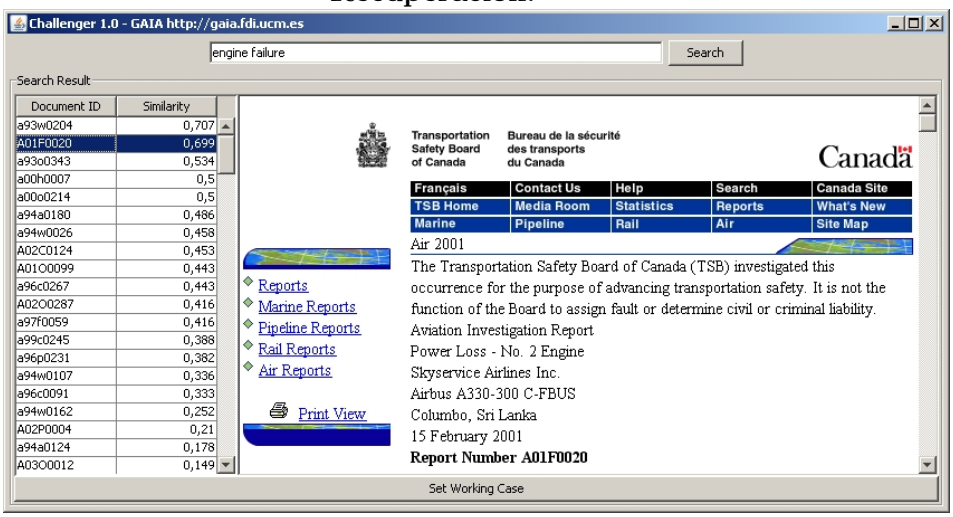


Figura 4.6: Recuperación en la aplicación Challenger

búsquedas locales focalizadas en una determinada sección.

La aplicación muestra una ventana de búsqueda donde el usuario puede introducir una consulta y recuperar los documentos más similares a su problema. Aquí se utiliza toda la funcionalidad del potente lenguaje de consultas de Lucene que incorpora comodines, intervalos y consultas difusas.

La Figura 4.6 muestra este paso. La parte superior de la ventana contiene la consulta y la parte izquierda una lista ordenada de documentos recuperados. Dichos documentos pueden ser visualizados en la parte principal de la ventana.

Una vez que el usuario ha elegido el documento más similar a su problema lo elige como caso de trabajo. Este documento servirá de base para realizar la adaptación mediante la substitución sus fragmentos.

Adaptación. En esta etapa se usa la estructura (organización en secciones) del caso de trabajo para buscar información relacionada que ayude al usuario en el proceso de adaptación. De esta forma, el sistema sólo busca información relacionada en la misma sección de cada documento. Aquí aparece el problema de gestionar documentos con diferente estructura y nombres de secciones. Por ello, es necesario relacionar las secciones de los distintos documentos. Esta aplicación utiliza una aproximación dividida en tres etapas:

1. Se relacionan secciones que tienen una única palabra en el título mediante WordNet. Por ejemplo, “Summary” y “Synopsis” son

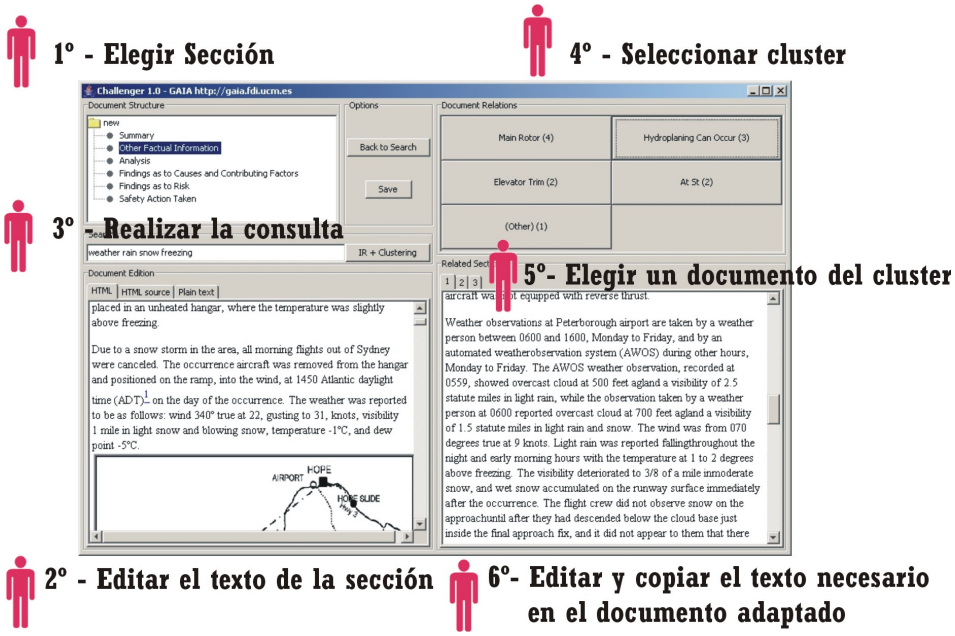


Figura 4.7: Adaptación en la aplicación Challenger

títulos muy frecuentes que representan la misma sección y pueden ser relacionados de esta forma.

2. Si el título contiene varias palabras, se aplica el método de extracción de raíces de jCOLIBRI y se comparan los títulos transformados. De este modo, pueden relacionarse secciones como: “Findings as to risks” y “Findings as to risk”.
3. Por último (y si los pasos previos han fallado), se utiliza un glosario definido por el usuario para encontrar otras relaciones. Por ejemplo: “Factual Information” y “Other Factual Information”.

Estas relaciones entre secciones se utilizan a continuación para encontrar información relevante en otros documentos. Así se aplica una búsqueda más localizada que evita la exploración de todo el documento (aunque también es posible realizar la búsqueda en todo el texto).

La Figura 4.7 muestra la ventana de adaptación. En la parte superior-izquierda se muestra la estructura del caso de trabajo. El usuario puede seleccionar cada sección de forma independiente y modificarla según sus necesidades. Para ello se utiliza el panel de edición situado en la parte inferior-izquierda. Este panel de edición incluye una visualización HTML, el código fuente HTML y la versión en texto plano. El usuario puede modificar el texto directamente (borrando o escribiendo) y también puede preguntar por secciones similares que pudieran ayudarle en

la adaptación. Hay dos formas de obtener dicha información: resaltando una parte del texto con el ratón o introduciendo algunas palabras clave en el cuadro de entrada de consultas. De este modo se introduce una consulta para obtener información relacionada que sea útil en la adaptación del texto. El proceso de búsqueda de esta información se divide en dos pasos:

1. Se utiliza Lucene para recuperar secciones de otros informes que sean relevantes a la consulta.
2. Se agrupa las secciones mostrando las etiquetas descriptivas devueltas por Carrot2.

Las etiquetas de los grupos obtenidos (en la parte superior-derecha de la Figura 4.7) ayudan al usuario a encontrar secciones relacionadas que contienen la consulta introducida. Cuando el usuario selecciona uno de los grupos, las correspondientes secciones de los documentos agrupados se muestran en la parte inferior-derecha de la ventana. El usuario puede reutilizar la información contenida en esas secciones para reescribir el caso de trabajo y adaptar su contenido a la nueva situación.

Como ejemplo de funcionamiento, la Figura 4.7 muestra al usuario editando la sección "Other Factual Information" donde se describen las condiciones metereológicas en el momento del accidente. Las condiciones metereológicas del caso base (el más similar a la consulta) son diferentes de las del escenario real que el usuario quiere reflejar en el informe. Así que el usuario necesita información sobre la forma de expresar dichas condiciones en otros documentos. De esta forma sabrá cómo plasmarlas en el texto que está generando.

Para obtener esta información el usuario busca algunos términos relacionados con el tiempo: "lluvia", "tormenta", etc. En este ejemplo el sistema devuelve 12 secciones relacionadas que se organizan en 5 grupos ("Main rotor","Hidro planning", "Elevator Trim",...). El sistema presenta las secciones recuperadas al seleccionar cada grupo y de esta forma el usuario puede reutilizar su contenido para redactar el nuevo informe.

Aunque el método expuesto se haya descrito inicialmente como un método *transformacional*, también podría encajar como método *generativo* donde la solución se construye desde cero en vez de derivarse. La adaptación constructiva (Plaza y Arcos, 2002) consiste en un proceso de búsqueda por el espacio de soluciones que utiliza información de los casos para guiar la búsqueda. Más específicamente, los casos se utilizan para generar y ordenar hipótesis. Nuestro método podría redefinirse como un método constructivo si no se copiara el caso recuperado y sólo se utilizara su estructura. El algoritmo de clustering se utilizaría para generar hipótesis que pertenecieran

al mismo grupo y el usuario estaría a cargo de ordenar y seleccionar dichas hipótesis.

La aplicación descrita en esta sección no permite evaluar específicamente los mismos parámetros que el recomendador de restaurantes (ver Sección 4.4.1). Sin embargo, pone de manifiesto la viabilidad e idoneidad de los métodos estadísticos en determinados escenarios. Siempre sería preferible utilizar los métodos clásicos de TCBR donde se extrae la información y se representa de forma estructurada. Pero si no se dispone de información sobre el dominio o es difícil extraer la información, la única opción son los métodos estadísticos. En el caso de la aplicación *Challenger* no se disponía de ningún conocimiento del dominio y los textos eran muy complejos tanto en semántica como en estructura. Por lo tanto, la aplicación propuesta es una aproximación muy viable de sistema de CBR Textual.

4.5 Conclusiones

Este capítulo hemos detallado todo el soporte que jCOLIBRI ofrece para desarrollar sistemas de CBR Textual (TCBR). El armazón permite implementar aplicaciones típicas de TCBR donde se utiliza una representación estructurada de los casos (TCBR Semántico), pero también aplicaciones donde no se pueda aplicar esta aproximación y es necesario manejar el texto directamente (TCBR Estadístico).

Las aplicaciones típicas de TCBR se denominan *semánticas* en jCOLIBRI porque intentan capturar y representar la información contenida en los textos. Dichas aplicaciones necesitan aplicar técnicas de Extracción de Información (IE) para obtener la información de los textos y almacenarla en los distintos atributos que conforman el caso. Este proceso de obtención de la información puede llegar a ser bastante complejo y, por ello, jCOLIBRI utiliza un modelo teórico (Lenz, 1998) que lo divide en diferentes capas. Nuestro armazón implementa dicho modelo de dos formas distintas según se explica en la Sección 4.1: utilizando algoritmos de Máxima Entropía (OpenNLP, 2008) y mediante la librería GATE (GATE, 2008). Estos métodos permiten realizar el procesamiento completo del texto que abarca desde la eliminación de palabras vacías y la extracción de raíces, expresiones y características, hasta la representación de la información obtenida en otros atributos del caso. A partir de ahí, se utilizarían los métodos clásicos de CBR para recuperar y adaptar los casos.

En el experimento descrito en la Sección 4.4.1 se demuestra que las aplicaciones de TCBR semántico obtienen mejores resultados que si únicamente se aplicaran métodos estadísticos de Recuperación de Información (IR) para obtener los casos similares a la consulta.

Sin embargo, existen determinados escenarios donde no es posible aplicar dichos métodos. Esto ocurre cuando no se dispone de conocimiento sobre el dominio que haga posible la extracción de información, o que dicho proceso sea demasiado complejo y propenso a errores por otros motivos como el tipo de lenguaje o estructura del texto. En estos casos la única solución posible es aplicar los métodos de procesamiento estadístico descritos en la Sección 4.2. Dichos métodos aplican algoritmos muy conocidos y testados en otras áreas de investigación para efectuar el proceso de recuperación, e incluso guiar al usuario en la adaptación de los textos. Cuando los métodos semánticos no son aplicables, los métodos estadísticos pueden ser muy útiles como se muestra en la aplicación descrita en la Sección 4.4.2.

Un uso muy importante de los métodos de CBR Textual es el procesamiento de consultas en lenguaje natural ya que es el mecanismo más cómodo para el usuario. Aplicando los métodos de TCBR junto con las técnicas para la incorporación de ontologías que se presentarán en el próximo capítulo es posible analizar una consulta textual y convertirla a una representación estructurada. La principal ventaja de esta aproximación es que el usuario no necesita conocer ni los atributos que conforman el caso ni los valores permitidos para cada uno de ellos.

En resumen, el trabajo sobre CBR Textual en jCOLIBRI permite implementar este tipo de aplicaciones de forma sencilla. Aunque se omiten los detalles técnicos del código (disponibles en los manuales), su diseño es completamente compatible y flexible. Esto permite elegir diferentes combinaciones de métodos (tanto semánticos como estadísticos). Además, la gestión de los casos, tipos de datos textuales y conectores se realiza de forma completamente transparente.

Capítulo 5

KI-CBR mediante ontologías

– Uhm... necesito otro tipo de clavos para poder probar bien mi martillo. ¡Haré también unas escarpias!

El núcleo de conocimiento de un sistema CBR está formado principalmente por la base de casos. Sin embargo, ese no es el único tipo de conocimiento con el que cuenta un sistema CBR. Existe otro tipo de conocimiento complementario entendido como cualquier información del sistema que no se encuentra recogida en forma de casos y que interviene en los distintos procesos del ciclo CBR. Como ejemplos de este conocimiento podríamos citar la información sobre similitud entre casos o la necesaria para adaptar sus soluciones. Obviamente, su importancia dentro del ciclo CBR dependerá de varios factores como el tipo de aplicación, la cantidad y calidad de la información contenida en los casos o la dificultad del dominio.

Durante varios años, la comunidad de CBR ha desarrollado diversos métodos para resolver los procesos del ciclo CBR en distintos dominios, para distintos tipos de aplicaciones, y haciendo uso de distintos tipos de conocimiento. Este conocimiento es necesario en mayor o menor medida dependiendo del tipo de aplicación CBR aunque siempre debe aparecer incluido en el sistema tal y como se mostraba en la Figura 2.2 (pág. 16) donde se introducía el ciclo CBR. Si se revisa dicho esquema puede comprobarse que además del conocimiento sobre los casos existe otro tipo de conocimiento del dominio que engloba la base de casos. Este conocimiento adicional utilizado por los sistemas CBR ha sido clasificado en términos de los siguientes contenedores (*knowledge containers* (Richter, 1998)):

- El conocimiento ontológico o vocabulario del dominio, que incluye la terminología utilizada en los casos, atributos y valores permitidos.

- Los casos descritos mediante esa terminología.

- Las medidas de similitud y otros criterios de recuperación. Este conocimiento pude estar representado de forma explícita –mediante jerarquías conceptuales que expliciten la relación entre los elementos del dominio, o de parámetros para pesar los atributos– o puede estar oculto dentro del algoritmo de recuperación.

- El conocimiento de adaptación o de transformación de la solución. Algunos sistemas usan conocimiento para verificar la utilidad de un caso elegido o adaptar su solución, por ejemplo aplicando los pasos que llevaron a dicha solución pero sobre la nueva consulta del usuario.

Todos los sistemas CBR comparten la representación explícita de los casos. Sin embargo el resto del conocimiento puede formar parte, de forma implícita, de los algoritmos que resuelven los procesos CBR o estar representado de forma explícita y declarativa como un depósito de conocimiento general sobre el dominio que es utilizado por los algoritmos. Realmente este conocimiento general del dominio que da soporte a los procesos CBR puede ser de muy distinta naturaleza, variar en el grado de completitud y representarse de diversas maneras: modelos causales, modelos de comportamiento, heurísticas, restricciones, modelos terminológicos o jerarquías conceptuales (Aamodt, 2001).

El desarrollo de métodos genéricos de CBR que puedan reaprovechar ese conocimiento entre distintas aplicaciones requiere modelos explícitos y reutilizables de conocimiento general que sirvan como soporte de los procesos CBR. Para solventar este inconveniente, en jCOLIBRI se propone una aproximación fundamentada en ontologías que permitan reutilizar el conocimiento entre las distintas aplicaciones CBR. Como se discutió en la Sección 2.2, las ontologías son un mecanismo muy eficiente para encapsular el conocimiento sobre el dominio de una aplicación CBR e incorporarlo a dicho sistema. De esta forma, el problema de la adquisición de conocimiento en los Sistemas Basados en Conocimiento se traslada a la existencia de una ontología que contenga la información necesaria por el sistema y que pueda ser reutilizada. Si no existiera dicha fuente de conocimiento sería necesario desarrollar una nueva ontología, aunque este proceso sí que aportaría una fuente de información reutilizable en futuros desarrollos. Además, la utilización de ontologías supone la incorporación a la aplicación de las posibilidades de razonamiento inherentes a dicho mecanismo de representación del conocimiento y que ya se expusieron al presentar las posibilidades de las Lógicas Descriptivas en la Sección 2.2.4 (clasificación, reconocimiento de instancias, etc.).

El calificativo de aplicación KI-CBR (aplicaciones CBR con conocimiento intensivo, del inglés Knowledge Intensive CBR) que da título a este capítulo enmarca toda la línea de trabajo encargada de estudiar las distintas vías de representación del conocimiento complementario a los casos aunque también

suele utilizarse para denominar a los sistemas que carecen de una base de casos amplia y necesitan complementar esa carencia de conocimiento a través de otras fuentes para mantener la eficiencia.

Las siguientes secciones irán desglosando las distintas formas de representar el conocimiento necesario por los sistemas KI-CBR mediante ontologías. En la próxima sección se ofrece una visión global de las diversas posibilidades para aprovechar la riqueza semántica de este tipo de representación de la información en el ciclo CBR. A continuación se irán detallando los distintos métodos de KI-CBR incluidos en jCOLIBRI siguiendo una organización basada en las principales etapas del ciclo CBR: persistencia de los casos, procesos de recuperación y métodos de adaptación.

5.1 El papel de las ontologías en jCOLIBRI

Las ontologías permiten el desarrollo de sistemas KI-CBR al reutilizar conocimiento que ya ha sido adquirido, conceptualizado e implementado en un lenguaje formal como las Lógicas Descriptivas. De este modo se reduce considerablemente el problema de adquisición de conocimiento y se complementa el conocimiento almacenado en los casos. La aproximación KI-CBR planteada en jCOLIBRI se basa en diseñar sistemas CBR que combinen el conocimiento de la base de casos y el conocimiento sobre el dominio representado en forma de ontologías. De esta forma, cuanto más conocimiento sea integrado en la aplicación CBR, más se aumentará la eficacia del mismo. Esta nueva familia de métodos KI-CBR es independiente de los otros métodos de CBR textual, conversacional o intensivo en casos que se incluyen en el armazón, aunque pueden ser combinados entre ellos.

La incorporación de las ontologías en el armazón permite la definición de métodos reutilizables que aprovechan el conocimiento semántico de las mismas. El proceso de incorporación de este conocimiento encaja perfectamente en el diseño de métodos de jCOLIBRI ya que éstos son independientes del dominio concreto de la aplicación y reciben la información sobre el mismo mediante sus distintos parámetros. De este modo se desacopla la funcionalidad y el conocimiento sobre el dominio tal y cómo se planteaba en los requisitos de los KBS presentados en la Sección 2.1. Esta aproximación contrasta con la mayoría de sistemas CBR que normalmente incluyen una enorme cantidad de conocimiento representado de forma implícita en sus distintas etapas de razonamiento. Además se posibilita su utilización en el proceso de adaptación ya que es la tarea que por su dificultad requiere más conocimiento.

Las ontologías utilizadas en jCOLIBRI están formalizadas mediante las Lógicas Descriptivas que se describieron en la Sección 2.2.4. Varios trabajos previos sugieren el uso de DLs para organizar, representar y recuperar casos

en sistemas CBR como MRL (Koehler, 1994), CATO (Ashley y Aleven, 1993), RESYN/CBR (Napoli et al., 1996), y un sistema de diagnóstico para la red de telefonía Francesa (Salotti y Ventos, 1998). Además, la implantación como estándar de facto del lenguaje OWL (más precisamente en su variante OWL-DL) hacen de esta elección la más adecuada para incorporar las ontologías en el armazón.

En este trabajo se propone la utilización de las ontologías para posibilitar distintos procesos de las aplicaciones CBR (Recio-García et al., 2006a):

1. Persistencia de los casos en ontologías. Las ontologías pueden servir para almacenar los casos mediante sus instancias o conceptos. Además, los conceptos utilizados para definir los casos pueden utilizarse como índices de la base de casos.

2. Como vocabulario para definir la estructura del caso. Esta posibilidad se fundamenta en que los conceptos, relaciones e individuos representan una descripción terminológica del dominio y, por lo tanto, puede utilizarse tanto si los casos se encuentran almacenados dentro de la ontología como si se encuentran en cualquier otro medio de persistencia.

3. Como la terminología necesaria para definir el vocabulario de la consulta. De esta forma el usuario puede expresar mejor sus requisitos al utilizar un vocabulario semánticamente más rico. Además, durante el cálculo de la similitud, la ontología permite rellenar el vacío terminológico entre la representación de consulta y la representación de la base de casos.

4. Para mejorar la eficiencia de los procesos de recuperación y similitud (González-Calero et al., 1999a; Salotti y Ventos, 1998; Napoli et al., 1996), adaptación (González-Calero et al., 1999b) y aprendizaje (Aamodt, 1990).

5. Por último, las ontologías permiten reutilizar el conocimiento entre diferentes sistemas CBR.

Los cinco puntos anteriores constituyen las aportaciones principales del uso de ontologías en los sistemas CBR que se describen este capítulo y servirán para estructurar los siguientes apartados. En ellos se describirán las ventajas del uso de ontologías en el proceso CBR y los distintos métodos desarrollados en nuestra plataforma para su explotación. Antes de empezar con la exposición también describiremos algunos detalles técnicos sobre su incorporación a la arquitectura de JCOLIBRI.

Para ilustrar los diferentes métodos basados en ontologías que se han desarrollado en el armazón utilizaremos una aplicación CBR denominada "Entrenador Personal". Esta aplicación se presenta en Recio-García et al.

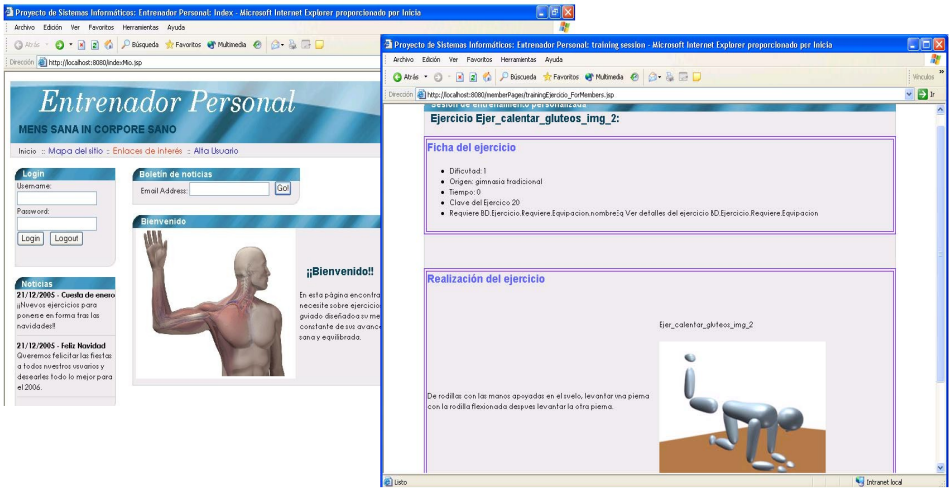


Figura 5.1: Captura del recomendador “Entrenador Personal”

(2006a) y su objetivo es la recomendación de sesiones de ejercicios dependiendo de una consulta que contiene información general sobre el usuario como la edad, sexo, estado de salud o si ha sufrido lesiones. Los casos de este sistema se almacenan una base de datos y una ontología. Los casos son descripciones de entrenamientos dirigidos a un perfil determinado de usuario que incluyen otra información multimedia como modelos 3D de los cuerpos que describen los ejercicios. El perfil del usuario y la información multimedia se almacena en la base de casos aunque la solución del caso –la sesión de ejercicios– se guarda en la ontología. Esta ontología estructura los ejercicios y representa el conocimiento necesario para adaptar el entrenamiento a un nuevo perfil de usuario. Este recomendador también utiliza la extensión web de jCOLIBRI para crear aplicaciones CBR con interfaz web, tal y como se muestra en la Figura 5.1. Como ya se comentó anteriormente, la ontología utilizada en este ejemplo está formalizada en el lenguaje OWL.

5.2 OntoBridge

El uso de conocimiento formalizado en un lenguaje declarativo como OWL conlleva el uso de interpretes o puentes que permitan gestionar ese conocimiento. El desarrollo de este componente intermedio no es una tarea sencilla ya que involucra tanto la carga y modificación de la ontología en memoria como la conexión con el razonador de Lógicas Descriptivas que realiza las inferencias. Aunque existen algunas librerías como JENA[1] que permiten la manipulación de ontologías, su complejidad supone un gran problema a la

[1] http://jena.sourceforge.net/

hora de desarrollar aplicaciones que la utilicen. Por lo tanto se hace imprescindible añadir una capa de abstracción más alta que facilite y simplifique la gestión de las ontologías.

Ese es el origen de la librería OntoBridge: un subproyecto de jCOLIBRI que permite manejar ontologías junto a varios razonadores de DLs de forma sencilla. Aunque inicialmente fuera un complemento de jCOLIBRI esta librería ha llegado a tener entidad propia debido a su utilidad a la hora de implementar aplicaciones de la Web Semántica tal y como se expone en Díaz-Agudo et al. (2008a). Esta librería también se ha puesto a disposición de la comunidad investigadora y se puede obtener través de la página web: http://gaia.fdi.ucm.es/projects/ontobridge/, donde se puede observar que cuenta con más de 700 descargas en sólo un año.

La principal virtud de OntoBridge es que oculta la gran complejidad de las APIs para el manejo de ontologías. Estas APIs para el acceso programático a ontologías son inherentemente complejas ya que las ontologías OWL son a su vez ontologías RDFS y RDF, por lo que cada elemento puede tratarse desde diferentes puntos de vista.

Aunque el API más extendida para la gestión de las ontologías es JENA, aunque existen otras de creciente popularidad como el OWLAPI[2]. En cualquier caso, OntoBridge es una abstracción permite utilizar ambas interfaces ocultando su complejidad al desarrollador. Respecto a los razonadores que pueden utilizarse mediante nuestra librería podemos destacar Pellet y FaCT++. Pellet[3] es un razonador implementado en Java, lo que implica dos características principales: su fácil integración en nuestros proyectos al compartir el mismo lenguaje y su poca eficiencia debido a las limitaciones de rendimiento del mismo. Por el contrario FaCT++[4] está implementado en el lenguaje C++ y por lo tanto presenta las características inversas: difícil integración pero gran rendimiento. De cualquier modo, OntoBridge permite conectar con casi cualquier razonador siempre que implemente el estándar *DIG*. Este estándar define un lenguaje de comunicación con los razonadores a través del protocolo *http*. Por lo tanto se garantiza una compatibilidad total independientemente del lenguaje de implementación del razonador.

Ontrobridge permite gestionar una o varias ontologías, e incorpora varios métodos para obtener, insertar, modificar o eliminar elementos de las mismas. Dichos elementos se dividen en conceptos, instancias y tipos de datos RDFS (como enteros, cadenas de texto, etc.). También se incorporan una serie de interfaces gráficas predefinidas para presentar fácilmente las ontologías al usuario, posibilitando también la elección de instancias de forma visual. Respecto a los procesos de razonamiento se incorporan métodos para el cálcu-

[2] http://owlapi.sourceforge.net/
[3] http://pellet.owldl.com/
[4] http://owl.man.ac.uk/factplusplus/

lo de diferentes inferencias como la clasificación de instancias y conceptos, el cálculo de profundidades o el cómputo del "Least Common Subsumer" (es decir, el concepto más específico que subsume a dos instancias) que es muy útil en los cómputos de similitud como se verá en la Sección 5.4. Por último, también se ofrece la posibilidad de utilizar el lenguaje SPARQL[5] para realizar consultas sobre la ontología mediante un esquema parecido a SQL.

Teniendo en cuenta que el diseño de OntoBridge se fundamentó en servir de apoyo tecnológico a jCOLIBRI está claro que su integración en la arquitectura del armazón es excelente. A grandes rasgos, jCOLIBRI incluye una clase que sirve de puente a la librería por medio del patrón de diseño *Singleton* (Gamma et al., 1995). Esta clase será accedida por los distintos métodos del armazón que necesiten utilizar la ontología.

5.3 Persistencia y representación de los casos mediante ontologías

La primera aportación de las ontologías al ciclo CBR que describiremos consiste en almacenar los casos en ellas. De esta forma es posible aprovechar las capacidades semánticas de este tipo de representación en las distintas etapas del CBR, principalmente en la recuperación y adaptación. Las ontologías son un medio de persistencia de la base de casos que sirve además para organizar y estructurar los casos a través de la jerarquía de conceptos y las propiedades que los relacionan.

En este tipo de persistencia los casos estarán representados como instancias de conceptos que están enlazadas a través de las propiedades de la ontología. Por lo tanto, los casos almacenados en la ontología tienen una estructura de grafo de instancias con una única instancia padre que identifica el caso. No hay que confundir esta estructura de grafo con la organización jerárquica de los conceptos. Los conceptos servirán para definir los tipos de los atributos de los casos (el rango) y las instancias serán los posibles valores de dichos atributos. Como una instancia suele estar enlazada a otras instancias a través de las propiedades, también obtenemos un grafo de instancias pero que constituye el valor de un atributo del caso. Para ilustrar esta estructura podemos fijarnos en la ontología mostrada en la Figura 5.2. En la parte superior de la figura aparece la organización jerárquica de los conceptos que definen el rango de un atributo (`Training, Exercises, Body, Leg, Arm,...`). La parte inferior muestra un grafo de instancias que sirve como valor al atributo `Solution` del caso. Dicho grafo se define a partir de la instancia `Training1` y las diferentes instancias enlazadas desde ella mediante propiedades (`Warm-up1, Main1, Cool-down1, Exer1, Exer2,...`).

[5] http://www.w3.org/TR/rdf-sparql-query/

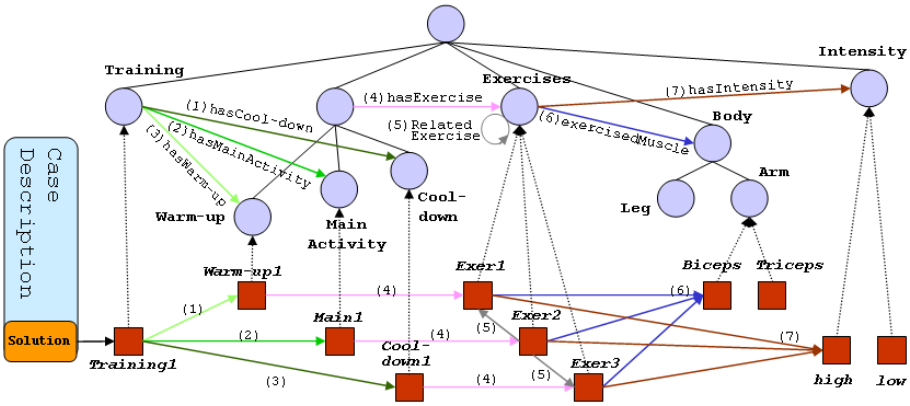


Figura 5.2: Esquema de persistencia mixta en ontologías

En jCOLIBRI hemos definido dos opciones distintas a la hora de manejar la persistencia de los casos en ontologías:

1. *Persistencia mixta*

   Consiste en almacenar algunos atributos del caso en la ontología, mientras que los restantes se guardan en un medio diferente (B.B.D.D., ficheros de texto, etc.). De esta forma, el caso que maneja jCOLIBRI tendrá algunos atributos que guardan un enlace con instancias de la ontología. Cuando un conector lea el caso desde el medio de persistencia elegido se realizará el enlace entre el atributo y la instancia de la ontología correspondiente. Un ejemplo de este esquema de persistencia lo encontramos en el recomendador de ejercicios. En este sistema la solución se almacena en la ontología y el resto del caso en una base de datos. Como la solución es un grafo de instancias bastante complejo, la representación del caso mediante las clases de jCOLIBRI solamente utiliza una referencia a la instancia padre desde uno de los atributos de la solución. El resto de instancias que pertenecen al grafo de la solución deberán ser accedidas por los métodos CBR a través de OntoBridge. La Figura 5.2 muestra un esquema de esta representación. En la parte izquierda aparece la representación del caso mediante las clases del armazón. Todo el caso está guardado en una base de datos, incluyendo aquí también el atributo que guarda la solución. Dicha columna de la base de datos contendrá los nombres de las diferentes instancias de la ontología que deben ser enlazadas al cargar los casos. Los conectores enlazarán automáticamente el atributo de la solución con la instancia correspondiente al realizar la carga en memoria. En la figura, las instancias a enlazar pertenecen al concepto *Training*. Por lo tanto, cada caso enlazará su atributo solución a cada una de las instancias de

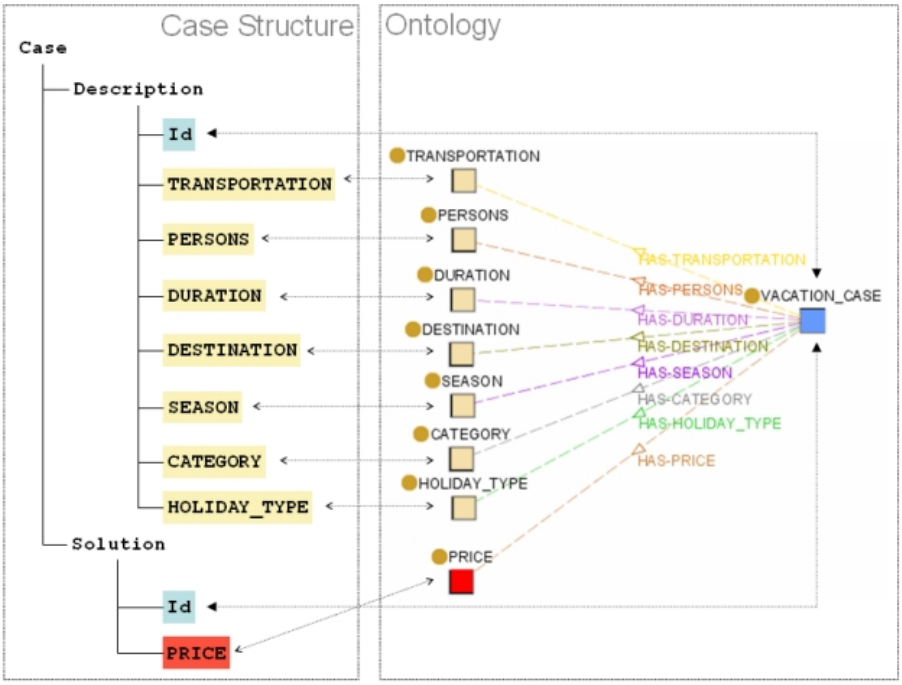


Figura 5.3: Esquema de persistencia completa en ontologías

ese concepto (aunque en la figura sólo se muestre una instancia por claridad).

De esta forma, la ontología permite que el atributo con la solución esté formado por todo una complejo árbol de instancias que enriquecen la expresividad del caso. Por ejemplo, en dicha figura, la solución del caso es una instancia del concepto *Training* llamada *Training1* que está formada por un calentamiento (Warm-up1), parte principal (Main1) y una vuelta a la calma (Cool-down1). Cada una de estas partes está compuesta por diversos ejercicios (*Exer1, Exer2 y Exer3*) que ejercitan cierto músculo y tienen un nivel de intensidad determinado. Siguiendo las propiedades que enlazan estas instancias y que se definen en los conceptos a los que pertenecen obtenemos una representación estructurada de la solución del caso mucho más rica semánticamente que un valor simple que pudiéramos asignar a dicho atributo.

2. *Persistencia completa*

El otro mecanismo de persistencia en ontologías consiste en mapear completamente el árbol de instancias a la representación del caso mediante las clases de jCOLIBRI. Esta opción permite reproducir exactamente la estructura de instancias en las clases de representación del

caso de jCOLIBRI. Para ello, el caso en deberá estar estructurado exactamente igual que las instancias de la ontología. En la Figura 5.3 se muestra un ejemplo de este tipo de persistencia. En este caso se está utilizando el recomendador de viajes que incorpora jCOLIBRI 2 como ejemplo y que se utiliza en su tutorial. Los casos del recomendador de viajes se componen de una descripción con distintos atributos que caracterizan un viaje y una solución con el precio. La ontología que alberga los casos contiene una serie de instancias que pertenecen al concepto *VACATION_CASE* y que están enlazadas a otras instancias que representan cada uno de los atributos del caso. En este tipo de mapeo, cada una de las instancias de la estructura del caso estará directamente enlazada por cada uno de los atributos del caso.

jCOLIBRI ofrece las dos opciones de persistencia anteriores, que deberán utilizarse dependiendo del tipo de recuperación o adaptación a realizar. Para enlazar una instancia con uno de los atributos del caso se utiliza el tipo de datos `Concept`. Este tipo de datos de jCOLIBRI representa una instancia de una ontología y puede utilizarse para definir el tipo de cualquier atributo. Este tipo de datos puede inicializarse a través de una simple cadena de texto que contenga el identificador de la instancia. El objeto `Concept` utilizará OntoBridge para acceder a la ontología y referenciar la instancia correspondiente.

Este proceso se realiza de forma transparente a los métodos y conectores, lo que posibilita la realización del primer tipo de mapeo donde los casos pueden estar almacenados en cualquier medio de persistencia como una base de datos o un fichero de texto. En dichos medios únicamente se tendrá que guardar el identificador de la instancia para cada atributo de tipo `Concept`. Al cargar los casos, cada atributo recibirá el identificador y enlazará con la instancia de la ontología que corresponda de forma completamente transparente al conector utilizado.

Para realizar el segundo tipo de mapeo se utiliza un conector específico para ontologías. Este conector necesita conocer además el nombre de las propiedades que enlazan las distintas instancias del caso para ir rellenando los atributos del caso. Por ejemplo, en la Figura 5.3 será necesario saber que para rellenar el atributo *Transportation* hay que seguir la propiedad *HAS_TRANSPORTATION* desde la instancia padre. Para obtener los casos, el conector recibe el nombre del concepto al que pertenecen las distintas instancias padres. De esta forma por cada instancia se genera un caso nuevo, donde el identificador es el nombre de la instancia padre y los atributos se rellenan siguiendo las propiedades que enlazan dicha instancia con el resto. Este mecanismo se ilustra en la Figura 5.4 donde se muestra una versión simplificada de la ontología del recomendador de viajes. En este ejemplo, los casos vienen definidos por el concepto *VACATION_CASE* y los diferentes atributos del caso por las instancias de los conceptos *PRICE*,

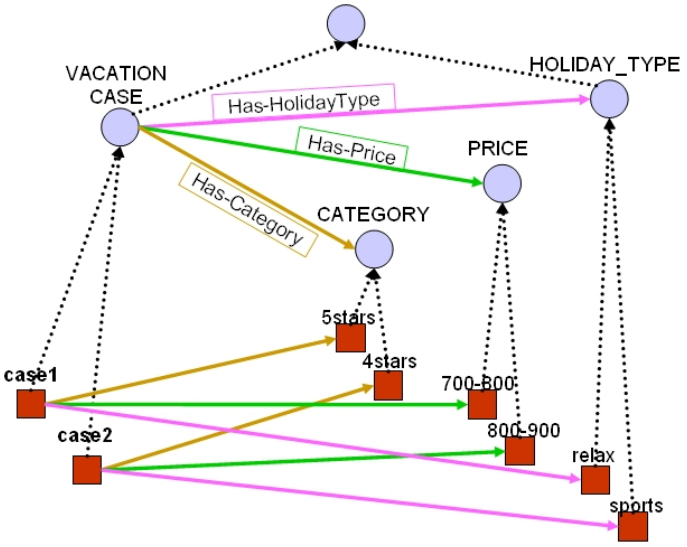


Figura 5.4: Ejemplo de base de casos almacenada completamente en una ontología

CATEGORY y *HOLIDAY_ TYPE*. Por cada una de las instancias de *VACATION_ CASE*: *Case1*, *Case2*, ... se generará un caso distinto cuyos atributos vendrán definidos por las diferentes instancias de los otros tres conceptos, ya que existen propiedades que los enlazan desde el concepto principal: *Has-price*, *Has-category* y *Has-HolidayType*. De este modo, los casos cargados en memoria tendrán serán: *Case1* compuesto por *5stars*, *700-800* y *relax*; y *Case2* compuesto por *4stars*, *800-900* y *sports*. Siguiendo el esquema general de los conectores, esta configuración se define mediante un archivo XML que el conector de ontologías recibe como parámetro.

5.4 Mecansimos de recuperación con ontologías

En el apartado anterior hemos visto las diferentes posibilidades de las ontologías como medio de persistencia. Sin embargo, la principal aportación de las ontologías a los distintos procesos CBR radica en el aprovechamiento del conocimiento almacenado en las mismas. El primero de estos procesos –la recuperación– se expone en esta sección, mientras que en las siguientes se expondrán los otros procesos que pueden beneficiarse del uso de ontologías.

Intuitivamente, es obvio que la jerarquía de conceptos e instancias de una ontología contiene conocimiento que puede ser utilizado para obtener la similitud entre la consulta y un caso. Esta utilidad de las ontologías para calcular la similitud ha sido utilizado en numerosas ocasiones por la comunidad CBR. Entre las distintas aproximaciones existentes hay que destacar

dos mecanismos de recuperación:

Recuperación basada en clasificación. Este método de recuperación utiliza las capacidades de clasificación de las Lógicas Descriptivas. Dentro de este grupo de métodos podemos distinguir dos técnicas distintas:

1. Recuperación basada en la clasificación de conceptos, donde la descripción de un concepto c_q se construye utilizando las restricciones que el usuario ha especificado en la consulta. Este concepto es entonces clasificado en la ontología y finalmente se recuperan todas sus instancias.
2. Recuperación basada en el reconocimiento de instancias, donde se construye una instancia i_q mediante la información especificada en la consulta del usuario. A continuación se incluye la instancia i_q en la ontología y se clasifica utilizando las capacidades de reconocimiento de instancias de las DLs. A continuación se obtienen los conceptos más específicos donde i_q ha sido clasificada y se recuperan todas las instancias de dichos conceptos.

Para entender este tipo de recuperación supongamos que en nuestro sistema de recomendación de viajes tuviéramos distintos conceptos que clasificaran las instancias que representan los viajes de acuerdo a ciertas propiedades. Por ejemplo, el concepto *viajes para esquiar* sería aquel cuyas instancia tuvieran como destino una estación de esquí, y el concepto *viajes a la playa* englobaría las instancias con una localidad de la costa en el destino. El usuario rellenará un formulario con sus preferencias –lo que incluirá el destino– y el sistema creará una instancia que represente dicha consulta. A continuación, esa instancia sería clasificada bien como perteneciente al concepto *viajes para esquiar*, o bajo el concepto *viajes a la playa*. A continuación, el sistema recuperaría las instancias del concepto donde haya sido clasificada ya que se supone serán más similares que las pertenecientes al otro concepto.

Recuperación computacional. Esta aproximación utiliza funciones de similitud numéricas para ordenar los casos con respecto a la consulta. Dichas funciones de similitud utilizan la posición de los distintos elementos dentro de la jerarquía conceptual de la ontología y son especialmente apropiadas para comparar casos estructurados.

Mientras que la recuperación basada en clasificación requiere que la base de casos esté completamente almacenada dentro de la ontología (esquema de persistencia completa), la recuperación computacional únicamente asume que una parte de la estructura del caso está representada en la ontología (esquema de persistencia mixta). Por lo tanto, la organización de la ontología puede utilizarse para calcular una similitud numérica de algunos atributos

del caso cuando sus valores se tomen de ella. Esta es la aproximación utilizada en jCOLIBRI ya que se adapta perfectamente al mecanismo de enlace de los atributos del caso con las instancias de la ontología que se realiza a través de los tipos de datos `Concept`.

En el trabajo de Bergmann y Stahl (1998) se propone un método para el cálculo de la similitud entre dos objetos donde se combinan dos factores: la similitud dentro de la clase o similitud por contenidos (*intra-class similarity*) y la similitud entre clases o por posición (*inter-class similarity*).

La componente de similitud por posición depende de la posición relativa de los dos instancias en la jerarquía conceptual del dominio, es decir, los conceptos a los que pertenecen. La componente de similitud por contenidos depende de las relaciones con otras instancias de las dos instancias comparadas. Estas instancias pueden a su vez ser estructurados si contienen relaciones con otras instancias, o simples si no presentan más relaciones. Ambas componentes se combinan usando una función de combinación como la media ponderada o simplemente el producto normalizado entre ambos valores. A continuación detallaremos cómo se han implementado estas medidas como métodos de jCOLIBRI.

5.4.1 Similitud por contenidos

jCOLIBRI incluye una función para calcular la similitud por contenidos. Para entender esta medida hay que tener en mente que una instancia se define por los conceptos a los que pertenece pero también por otras instancias con las que se relaciona a través de propiedades. Por lo tanto, al comparar dos instancias hay que medir su distancia dentro de la jerarquía de conceptos y la similitud entre las instancias con que se relacionan cada una de ellas. Por lo tanto, la similitud por contenidos debe recorrer este grafo de instancias relacionadas e ir comparando cada una de las instancias comunes.

La similitud por contenidos funciona recursivamente sobre la estructura de las instancias comparadas i_1 e i_2. Para ello, primero se calcula la similitud por posición de ambas y se obtiene la lista de propiedades que enlazan cada instancia con otras instancias. De este modo se obtienen las listas de propiedades $P_1 = \{p \mid \exists\ p(i_1)\}$ y $P_2 = \{p \mid \exists\ p(i_2)\}$. A continuación, para cada propiedad común en ambas listas ($p \in \{P_1 \cap P_2\}$) se obtienen las instancias que aparecen en el destino de cada una: $d_1 = p(i1)$ y $d_2 = p(i2)$ y se aplica de nuevo la función de similitud sobre ellas. Los valores devueltos al comparar las distintas propiedades de la instancia se combinan utilizando la media aritmética. La siguiente fórmula resume el proceso de comparación:

$$intrasim(i_1, i_2) = \frac{1}{2}\left(intersim(i_1, i_2) + \frac{\sum_{p \in P_1 \cap P_2} intraclass(p(i_1), p(i_2))}{\|P_1 \cap P_2\|}\right)$$

La recursión termina cuando dos instancias no pueden ser comparadas por contenido porque son hojas del grafo. En este caso sólo se aplicará la función de similitud por posición. Además hay que tener en cuenta que una instancia puede tener propiedades multivaloradas, es decir, varias relaciones del mismo tipo con distintas instancias. Por ejemplo, un viaje podría tener diversos destinos y cada instancia que representara un destino podría estar enlazada con otras instancias de la ontología que determinaran la duración de la estancia en dicho destino, el coste de esa parte del viaje, etc. Esta multiplicidad en los grafos de instancias hace que su comparación sea un problema NP completo ya que es necesario calcular las distintas combinaciones posibles para obtener el valor máximo de similitud. Como esta posibilidad es inviable a la hora de utilizar el método en escenarios reales, jCOLIBRI utiliza un algoritmo voraz que realiza la comparación con coste lineal aunque puede obtener como resultado un máximo local en vez de absoluto.

5.4.2 Similitud por posición

Las funciones de similitud por posición dependen de la posición que ocupan las dos instancias comparadas en la taxonomía conceptual del dominio, es decir, depende de los conceptos a los que pertenecen ambas instancias. La componente de similitud por posición no tiene en cuenta explícitamente las relaciones de las instancias con otras instancias que sí se consideran en la similitud por contenido. Sin embargo, las relaciones intervienen de forma implícita ya que la pertenencia o no de una instancia a un concepto depende también de cómo se relacione con otras instancias. Esto se debe a que en la definición del concepto se caracterizan las relaciones que las instancias de este concepto tendrán con otras instancias. Por ejemplo, al explicar la recuperación por clasificación se mencionaban los conceptos *viajes para esquiar* y *viajes a la playa* que se definían a partir de la relación *destino* de las instancias. Imaginemos que comparáramos una instancia que perteneciera al primer concepto y otra que perteneciera al segundo. Si nos basamos en la distancia de ambos conceptos dentro de la ontología también estaríamos utilizando información sobre sus relaciones de forma indirecta.

La mayoría de las funciones de similitud por posición implementadas en jCOLIBRI se basan en computar la posición que ocupa en la jerarquía el concepto más específico que contiene a las dos instancias comparadas. Este concepto se conoce como el "Least Common Subsumer" (LCS) y como ya vimos en la Sección 5.2, la librería OntoBridge permite obtenerlo directamente. En el cálculo de estas funciones hay que tener en cuenta que una instancia puede pertenecer a distintos conceptos de la ontología que se encuentren clasificados en diferentes ramas de la taxonomía. De este modo, el método que calcula el LCS puede devolver varios conceptos. Esta propiedad complica bastante el cálculo del LCS, sin embargo dicha complejidad es transparente al usuario gracias a OntoBridge.

A continuación detallamos las distintas funciones de similitud por posición incluidas en jCOLIBRI que se adaptan de las definidas en Díaz-Agudo (2002).

Función de similitud profundidad básica. Esta función computa la similitud en base a la relación existente entre la profundidad del concepto más específico de todos los que contienen a los dos individuos comparados y la profundidad máxima de la jerarquía.

$$\text{fdeep_basic}(i_1, i_2) = \frac{\text{máx}(\text{prof}(\text{LCS}(i_1, i_2)))}{\underset{C_i \in CN}{\text{máx}}(\text{prof}(C_i))}$$

Donde:

- CN es el conjunto de conceptos de la ontología.
- $\text{LCS}(i_1, i_2)$ es el conjunto de los conceptos más específicos que contienen a las dos instancias comparadas.
- $\text{prof}(x)$ es la profundidad en la ontología.

Función de similitud profundidad. De forma similar a la anterior, esta función computa la similitud según la relación que exista entre la profundidad del concepto más específico que contiene a los dos individuos comparados y la del individuo más profundo de los dos.

$$\text{fdeep}(i_1, i_2) = \frac{\text{máx}(\text{prof}(\text{LCS}(i_1, i_2)))}{\text{máx}(\text{prof}(i_1), \text{prof}(i_2))}$$

Función de similitud coseno. Se basa en la función de similitud entre conceptos definida en González-Calero (1997) e inspirada en el modelo del espacio vectorial. Esta medida se basa en representar cada concepto mediante un vector de propiedades y calcular la similitud entre dos conceptos como el resultado de aplicar una cierta función a los vectores que los representan. Cada concepto se representa mediante un vector con tantas componentes como conceptos haya en la ontología y cuyas valores son 1 o 0 dependiendo de si el concepto representado es o no un subconcepto del concepto asociado a esa componente. En dicho trabajo se propone una función de similitud entre dos instancias basada en el coseno del ángulo formado por los vectores que representan a los conceptos a los que pertenecen las instancias y se demuestra que esa función de similitud se computa mediante la siguiente expresión:

$$\text{cosine}(i_1, i_2) = \frac{\left|\left(\bigcup_{d_i \in \text{t}(i_1)} (\text{super}(d_i, CN))\right) \bigcap \left(\bigcup_{d_i \in \text{t}(i_2)} (\text{super}(d_i, CN))\right)\right|}{\sqrt{\left|\bigcup_{d_i \in \text{t}(i_1)} (\text{super}(d_i, CN))\right|} \cdot \sqrt{\left|\bigcup_{d_i \in \text{t}(i_2)} (\text{super}(d_i, CN))\right|}}$$

Donde:

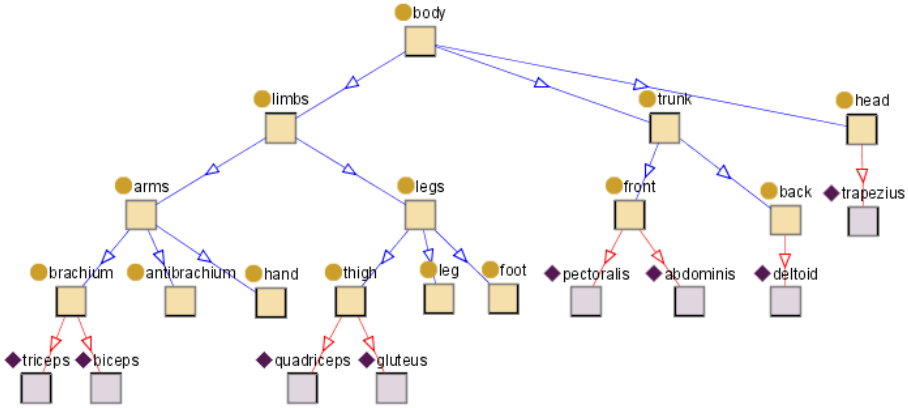
- $\text{super}(c_i, CN)$ es el conjunto de superconceptos de c en C, siendo C un conjunto de conceptos dados y c un concepto que pertenece a dicho conjunto.
- $t(i)$ es el conjunto de conceptos al que pertenece la instancia i.

Función de similitud detalle. En esta medida, la similitud por posición de dos instancias se define en función del número total de conceptos a los que pertenece una instancia. De esta forma, cuanto mayor sea el número de conceptos mayor será el nivel de detalle o de información con el que se describen y por lo tanto la similitud también será mayor.

$$\text{detail}(i_1, i_2) = 1 - \frac{1}{2 \cdot \left|\left(\bigcup_{d_i \in \text{t}(i_1)} (\text{super}(d_i, CN))\right) \bigcap \left(\bigcup_{d_i \in \text{t}(i_2)} (\text{super}(d_i, CN))\right)\right|}$$

Para ilustrar el comportamiento de estas medidas de similitud la Figura 5.5 muestra el resultado de aplicarlas a la ontología utilizada en la aplicación *Entrenador Personal*. Como puede observarse, la similitud entre instancias cercanas dentro del árbol de conceptos será siempre mayor que la similitud entre instancias con mayor distancia entre ellas. Cada una de estas medidas tiene sus propias características y deberá elegirse de acuerdo a la ontología que esté siendo utilizada. Por ejemplo, las funciones de similitud por profundidad básica y coseno devuelve un valor de 1 entre instancias hermanas aunque estas sean diferentes y este hecho puede tener una gran repercusión en el proceso de recuperación.

Las medidas de similitud aquí presentadas permiten aplicar técnicas de recuperación basadas en el algoritmo k-NN. Como ya se explicó en la Sección 2.1.1 este algoritmo utiliza funciones de similitud locales para comparar los atributos sencillos y medidas de similitud globales que combinan los valores devueltos por las medidas locales. Las funciones de similitud basadas en



	fdeep_basic	fdeep	cosine	detail
sim(triceps,biceps)	1	4/5	1	7/8
sim(pectorals,abdominis)	3/4	3/4	1	5/6
sim(triceps,abdominis)	1/4	1/5	$1/\sqrt{12}$	1/2
sim(triceps,cuadriceps)	1/2	2/5	1/2	3/4
sim(pectorals,deltoid)	1/2	1/2	2/3	3/4

Figura 5.5: Ejemplo de medidas de similitud basadas en ontologías

ontologías incorporadas a jCOLIBRI actúan como medidas de similitud locales dentro del algoritmo k-NN del armazón. De esta forma es posible asignar una de las medidas anteriores para calcular la similitud de cualquier atributo de tipo `Concept` presente en la estructura del caso.

Tanto en la sección anterior como en ésta se ha mencionado que el tipo de datos `Concept` permite utilizar instancias de una ontología dentro de la estructura del caso. Además de las ventajas algorítmicas para la recuperación y el cómputo de similitud, la inclusión de valores procedentes de ontologías en los atributos del caso implica una mejora considerable en la expresividad de los mismos y de las posibles consultas al sistema CBR. A continuación describiremos qué repercusiones tiene la incorporación de ontologías como el vocabulario utilizado para definir los distintos valores de los atributos en casos y consultas.

5.5 Las ontologías como el vocabulario del sistema CBR

Las ontologías pueden utilizarse dentro de una aplicación CBR para definir los posibles valores de los atributos. Para ello, se utiliza la clasificación jerárquica en la ontología de las instancias que sirven de valor en los atributos del caso. Este uso de la ontología como vocabulario del sistema puede aplicarse de diferentes maneras dependiendo de si se utiliza para definir los casos o las

consultas que el usuario puede realizar al sistema. A continuación detallaremos ambas aproximaciones y su repercusión en la expresividad del sistema según se presenta en Díaz-Agudo et al. (2007b); Recio-García et al. (2006a).

Respecto al *vocabulario del caso*, existe una aproximación directa que consiste en el uso de la ontología del dominio al estilo del paradigma de orientación a objetos: los conceptos son los tipos, y las instancias son los valores permitidos para dichos atributos. Además, las relaciones entre los conceptos pueden servir para definir la estructura del caso de forma que un atributo esté compuesto por otros atributos y esta relación de composición se obtenga a partir de las propiedades de la ontología.

Esta aproximación es la que se utiliza al asignar el tipo `Concept` a un atributo de un caso en jCOLIBRI. Al realizar esta asignación se definen los valores permitidos para dicho atributo como las diferentes instancias de un determinado concepto de la ontología. Por ejemplo, imaginemos que estuviéramos utilizando la base de casos representada en la ontología de la Figura 5.4. En este escenario, el caso que manejaría jCOLIBRI estaría compuesto por tres atributos de tipo `Concept`: *categoría*, *precio* y *tipoVacaciones*. Si estableciéramos que el atributo *categoría* estuviera definido por el concepto `CATEGORY`, los únicos valores permitidos serían `4stars` y `5stars`.

También pueden existir aplicaciones donde los valores de un atributo `Concept` pueda ser cualquier instancia de la ontología. Para ello, deberíamos definir dicho atributo mediante el concepto padre de la ontología. De esta forma cualquier instancia sería un valor permitido para el atributo.

Este uso de la ontología como vocabulario de los casos también aparece, aunque más restringido, al utilizar el conector con ontologías. En este caso, siempre se indica a qué concepto pertenecen las instancias que sirven de valores para cierto atributo y este se encargará de enlazarlas al leer los casos.

Aunque en jCOLIBRI el uso de la ontología como vocabulario del caso esté ligado a que los valores de los atributos se encuentren en dicha ontología, esto no tiene por que ser siempre así. Por ejemplo, en la Figura 5.5 podríamos utilizar el concepto *body* para definir un atributo cuyos valores fueran *biceps*, *quadriceps*, *abdominis*,... Sin embargo, dichos valores no tendrían por qué ser instancias de la ontología sino simples cadenas de texto. En este caso, la aplicación CBR deberá comprobar que dichos textos aparecen como instancias del concepto *body*.

Respecto a la consulta, existen dos opciones para definir el vocabulario utilizado:

- Utilizar exactamente el mismo vocabulario que aparece en los casos. Es decir, utilizar únicamente las instancias que pertenecen al concepto asignado a cada atributo.

- Utilizar toda ontología como el vocabulario de la consulta. Esta opción permite utilizar consultas más expresivas ya que el usuario puede expresar mejor sus requisitos si se utilizan las diferentes instancias de la ontología. El aumento en la expresividad se debe a que la ontología puede definir instancias que no aparezcan en la base de casos. Estas instancias pueden incluirse en la consulta y luego utilizar las medidas de similitud basadas en ontologías para realizar la recuperación. De esta forma se permite enlazar la terminología de los casos y la terminología con que el usuario rellena la consulta.

Para entender la utilidad de la ontología para mejorar la expresividad de la consulta utilizaremos el ejemplo del *Entrenador Personal*. Imaginemos que la aplicación maneja una base de casos donde se ha definido un tipo enumerado *body* cuyos valores permitidos fueran los músculos: *biceps*, *pectoralis*, etc. Supongamos que $case_i$ es un caso que contiene ejercicios donde se utiliza el músculo *biceps*. En el sistema no se desea restringir el vocabulario de la consulta a los valores del enumerado y permitir expresiones más genéricas como por ejemplo:

- Consulta 1: "I want a *biceps* exercise"
- Consulta 2: "I need to exercise my *arm*"
- Consulta 3: "I would like to train my *limbs*"

Para estas tres consultas y utilizando la ontología presentada en la Figura 5.5 se encontraría $case_i$ como un candidato posible[6] aunque en las dos últimas consultas no se utilice directamente el mismo vocabulario que en los casos. La recuperación de este caso se realizará por medio de una de las medidas de similitud basadas en ontologías que se detallaron en la Sección 5.4.

Esta misma característica es la que se utiliza para mejorar las consultas en el recomendador de viajes presentado en la Sección 4.3. En este ejemplo, los valores utilizados para describir la temporada eran cada uno de los meses mientras que el usuario podía utilizar directamente términos como "verano" para describir sus preferencias. Utilizando una ontología sobre viajes, el sistema cambiaba automáticamente ese valor por "Junio, Julio, Agosto" que son los valores admitidos por el atributo.

El uso de ontologías para definir el vocabulario de casos y consultas junto con las distintas posibilidades de persistencia y cálculo de similitud constituyen una aportación muy importante para la etapa de recuperación en aplicaciones KI-CBR. Sin embargo, esta aportación al ciclo CBR no termina

[6]En este ejemplo se realiza la suposición de que existen las instancias *arm* y *limb* correspondientes a los conceptos con el mismo nombre.

ahí ya que las ontologías también pueden jugar un papel muy importante en la fase de adaptación según se explicará en la próxima sección.

5.6 Adaptación basada en ontologías

El aspecto clave del CBR como paradigma de resolución de problemas es que las soluciones a nuevos problemas son generadas aplicando las soluciones recuperadas de problemas similares previamente resueltos, en lugar de partir de cero. En muchas ocasiones, las soluciones así obtenidas son adecuadas y se pueden aplicar directamente. En otros casos, están próximas a la solución requerida pero no lo suficiente, por lo que resulta necesario adaptarlas. Aunque el papel de la adaptación es fundamental, esta etapa es en cierta forma el "Talón de Aquiles" del ciclo CBR ya que está muy poco estudiado y no existen técnicas estándar.

La adaptación de los casos es una tarea que necesita mucho conocimiento sobre el dominio por lo que muchos sistemas CBR han optado por incluir una enorme cantidad de conocimiento de adaptación en forma de reglas. La principal desventaja de esta aproximación es que dicho conocimiento es difícilmente reutilizable entre aplicaciones. Por este motivo, en jCOLIBRI se propone diversos métodos de adaptación basados en ontologías. Las ontologías nos permiten reutilizar el conocimiento sobre el dominio necesario para la adaptación de las soluciones y además nos ofrecen ciertas capacidades de inferencia que pueden ser útiles en ese proceso.

Para presentar los métodos de adaptación con ontologías incluidos en nuestro armazón los clasificaremos en los dos grupos principales de técnicas que aparecen en la bibliografía (Kolodner, 1993; Watson, 1997; Plaza y Arcos, 2002):

- La *adaptación generativa* construye una solución para el nuevo problema. La generación de dicha solución puede llevarse a cabo de dos maneras distintas:
  - Las técnicas de *adaptación derivacional* vuelven a aplicar el proceso de obtención de la solución original utilizando las nuevas características del problema. Para ello, el sistema debe incluir en la solución de los casos trazas sobre el proceso que condujo a dicha solución. Estas trazas se utilizarán para generar la nueva solución repitiendo todo el proceso de generación pero con la información del problema actual.
  - Las técnicas de *adaptación constructiva* utilizan las soluciones de los casos más similares para generar una nueva solución. La diferencia con las técnicas derivacionales radica en que la información

utilizada para generar la solución no son las trazas sobre el proceso que condujo a su obtención. En vez de guardar dicha información intermedia se utilizan directamente las soluciones de los casos más similares para construir una nueva solución a partir de ellas.

- La *adaptación transformacional* constituye el otro gran grupo de técnicas. En este caso se aplican operadores o fórmulas de adaptación a la solución de los casos que sirven para transformarla y adaptarla al problema actual.

En esta clasificación se remarca el origen de la información necesaria para construir la solución. Por un lado, la adaptación derivacional utiliza trazas del proceso de generación (decisiones, opciones, justificaciones, etc.) mientras que la adaptación transformacional y la adaptación constructiva se basan directamente en el conocimiento almacenado en la solución. Es en estas dos últimas técnicas donde la inclusión de las ontologías se presenta especialmente beneficiosa ya que permiten almacenar y estructurar el conocimiento sobre la solución de los casos. Por lo tanto, los métodos de jCOLIBRI que presentaremos a continuación se dividen en métodos de adaptación transfomacional (TA del inglés Transformational Adaptation) y métodos de adaptación constructiva (CA del inglés Constructive Adaptation).

5.6.1 Adaptación Transformacional

La aproximación transformacional aplica una serie de operandos sobre la solución de un problema similar para adaptarlo a la situación actual. Realmente no se aplican directamente sobre la solución del problema recuperado sino sobre una copia de la misma. En jCOLIBRI nos hemos basado en esquema transformación presentado en González-Calero et al. (1999b) donde se aplican una serie de substituciones de los elementos que forman la solución. Estas substituciones se basan en una serie de *instrucciones* que guían el proceso y que son definidas por el desarrollador. Además de substituciones también se permite incluir instrucciones de adición y borrado de elementos.

Este proceso de adaptación comparte algunas ideas con el trabajo descrito en Mille et al. (1996) donde se presenta un armazón para unificar los mecanismos de adaptación. En este trabajo, la adaptación se define como substituciones de elementos origen por elementos destino (incluyendo en las substituciones la adición y borrado de elementos). Además, se propone una idea muy interesante que consiste en aplicar el proceso de substitución de forma recursiva siguiendo las relaciones entre los distintos elementos que forman la solución. Esta idea encaja muy bien en nuestra aproximación basada en ontologías ya que también se maneja una representación estructura de las instancias que forman la solución, tal y como puede observarse en la Figura 5.2. En esta figura se muestra la representación de las soluciones en la aplicación Entrenador Personal. Cada solución se identifica por una instancia

del concepto *Training* que a su vez se compone de un calentamiento (*Warm-up*), parte principal (*Main Activity*) y la vuelta a la calma (*Cool-Down*). Cada una de estas partes estará compuesta por una serie de ejercicios (instancias de *Exercises*) que a su vez presentan una serie de propiedades como la intensidad (*Intensity*) y la parte del cuerpo ejercitada (instancias de los subconceptos de *Body*).

El método de adaptación transformacional utilizado en el Entrenador Personal se divide en los siguientes pasos:

1. Obtener la lista L de elementos en la solución que necesitan ser adaptados. Para ello se sigue el *camino de relaciones* (instrucciones) que identifican cada elemento. Por ejemplo, si se desea substituir un ejercicio del calentamiento habría que definir la secuencia de relaciones que conducen desde la instancia de *Training* que define la solución hasta la instancia concreta que define el calentamiento y desde allí la relación hasta dicho ejercicio. Esta cadena de relaciones se define en base a las instancias y propiedades que relacionan los distintos elementos de la solución.

2. Substituir cada elemento de L por otro elemento más apropiado. Para encontrar esos substitutos se utiliza una *búsqueda especializada* que se aprovecha de las ventajas ofrecidas por las Lógicas Descriptivas como se describirá a lo largo de esta sección.

3. Si existen dependencias entre los elementos substituidos, se modifican dichas dependencias apropiadamente. Por ejemplo, si cambiamos la parte del cuerpo utilizada en un ejercicio habrá que modificar también el ejercicio de la parte del calentamiento correspondiente.

La búsqueda especializada, según se describe en Kolodner (1993), es una forma de encontrar los candidatos a la substitución donde se utilizan una serie de instrucciones que indican los criterios para identificar dichos candidatos. En esta aproximación las instrucciones se definen manualmente por el desarrollador mediante el lenguaje que expondremos a continuación.

El lenguaje utilizado para definir las instrucciones de substitución se basa en operadores simples compuestos por una condición y una acción a realizar si dicha condición se cumple. Aunque existen en la Web Semántica algunos lenguajes de reglas que podrían aplicarse para definir los operadores como RuleML[7] o SWRL[8] su expresividad basada en claúsulas de Horn los hacen demasiado complejos para los objetivos de nuestro método. Por lo tanto se opta por una sintaxis basada en XML que simplifique la complejidad de las reglas. Estas reglas se componen de tres partes:

[7] http://www.ruleml.org/
[8] http://www.w3.org/Submission/SWRL/

```
- <adaptationRule>
  - <instanceLocation>
      <property>hasMainActivity</property>
      <property>hasExercise</property>
    </instanceLocation>

  - <condition>
    - <queryAttribute-Slot_Condition>
        <queryAttribute>Lesion</queryAttribute>
        <comparationOp>equals</comparationOp>
      - <slot>
          <property>exercisedBody</property>
          <concept>Body</concept>
        </slot>
      </queryAttribute-Slot_Condition>
    </condition>

  - <anyOtherModification>
    - <slot>
        <property>exercisedMuscle</property>
        <concept>Body</concept>
      </slot>
    - <condition>
      - <queryAttribute-Slot_Condition>
          <queryAttribute>Lesion</queryAttribute>
          <comparationOp>notEqual</comparationOp>
        - <slot>
            <property>exercisedMuscle</property>
            <concept>Body</concept>
          </slot>
        </queryAttribute-Slot_Condition>
      </condition>
      <followDependence>relatedExercise</followDependence>
    </anyOtherModification>

  </adaptationRule>
```

Figura 5.6: Ejemplo de regla de adaptación en ontologías

1. Identificación de la instancia a substituir (I) mediante una cadena de propiedades de la ontología.

2. Condición que debe cumplirse para ejecutar la instrucción. Se permiten tres tipos de condiciones:

   - Que un atributo de la consulta sea igual a uno de los elementos que forman la solución (i.e. alguna de las instancias que forman la solución).
   - Que un atributo de la consulta tenga un valor específico.
   - Que alguna de las instancias de la solución pertenezca a un determinado concepto.

3. Operador de adaptación. Substituye una instancia por otra que puede identificarse directamente, mediante condiciones que debe cumplir o mediante su pertenencia a determinados conceptos. Estos operadores de adaptación pueden aplicarse recursivamente a distintas instancias de la solución que estén relacionadas por una determinada propiedad.

Es en las condiciones y los operadores de adaptación anteriores donde se aprovechan los mecanismos de inferencia de las Lógicas Descriptivas como el reconocimiento de instancias.

Para ejemplificar este proceso de adaptación utilizaremos de nuevo el Entrenador Personal. Imaginemos que un usuario presenta una lesión en el brazo y el entrenamiento de la base de casos que mejor se adapta a su perfil contiene un ejercicio que utiliza esa parte del cuerpo. Para adaptar la solución debería definirse una regla como la presentada en la Figura 5.6. Dicha regla identifica los ejercicios de la parte principal del entrenamiento y cambia la parte del cuerpo ejercitada por otra distinta a la que aparezca en el atributo *Lesion* de la consulta. Además, dicha regla se aplica recursivamente siguiendo la relación *relatedExercise* para encontrar los ejercicios del calentamiento y

vuela a la calma que relacionados. La descripción de esta regla en un lenguaje informal sería:

```
INSTANCELOCATION:    hasMainActivity/hasExercise
CONDITION:           exercisedMuscle/Body == Lesion
ANYOTHERINSTANCEOF:  SLOT:                exercisedMuscle/Body
                     CONDITION:           exercisedMuscle/Body != Lesion
                     FOLLOWDEPENDENCIES:  relatedExercise
```

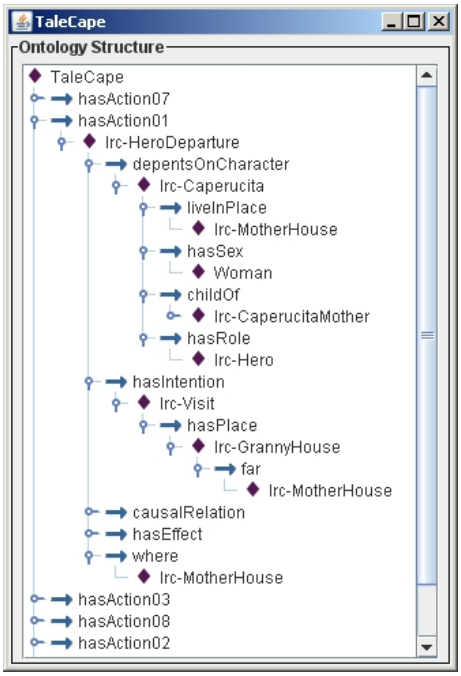
Para terminar con el ejemplo, la siguiente regla ajusta el nivel de intensidad de los ejercicios de acuerdo con el nivel de forma física que el usuario ha indicado en la consulta. Al igual que la regla anterior, las modificaciones se propagan siguiendo una propiedad de la ontología:

```
INSTANCELOCATION:    hasMainActivity/hasExercise
CONDITION:           HealthState == low
DIRECTMODIFICATION:  SLOT: hasIntensity/Intensity
                     INSTANCENAME:        low
                     FOLLOWDEPENDENCIES:  relatedExercise
```

Aunque el lenguaje anterior demostró ser suficiente para una aplicación como el Entrenador Personal esto puede no cumplirse en otro tipo de aplicaciones. Una posibilidad para solventar dicha carencia podría consistir en ampliar el lenguaje de reglas. Sin embargo, esta opción complicaría en exceso la definición de las mismas y dificultaría enormemente su manejo por parte del método de adaptación. Dado este escenario, se ha optado por desarrollar otro método de adaptación que sigue una aproximación opuesta donde se simplifica radicalmente la definición de las operaciones de substitución. Al realizar dicha simplificación se amplia enormemente la generalidad del método permitiendo aplicarlo en numerosas situaciones.

Esta simplificación consiste en utilizar las medidas de similitud presentadas en la Sección 5.4 para encontrar los candidatos a ser substituidos. Más específicamente se utiliza la medida de similitud por contenidos (intra-class) para comparar las distintas instancias y así determinar cuales son los mejores substitutos. De esta forma el desarrollador únicamente debe indicar qué instancias deben ser añadidas, borradas o substituidas en la solución. Para añadir se buscan instancias de la solución que pudieran contener propiedades compatibles en rango con la instancia a insertar. El proceso de borrado es directo: si se encuentra la instancia indicada por el desarrollador se elimina directamente incluyendo también todo el subárbol de dependencias. Por último, por cada instancia a substituir se calcula su similitud con el resto de instancias de la ontología mediante la medida intra-class y se cambia por la que presente un valor más alto.

Este método de adaptación transformacional se presenta en Díaz-Agudo et al. (2008b) donde se aplica al dominio de la generación de cuentos. En dicho



Título	Nº Instancias	Complejidad
La Adivina	22	3
Caperucita Roja	63	6
Cenicienta	66	4
El Patito Feo	103	4
El Pez de Oro	184	5
La Zorra, la Liebre y el Gallo	106	4

Figura 5.7: Ejemplo de cuento representado con ontología

trabajo se realiza una comparativa de los métodos de adaptación transformacional y constructiva según detallaremos en las próximas secciones. Para realizar dicha comparativa se utilizó una ontología sobre cuentos basada en la teoría de Propp (Propp, 1968). Dicha teoría propone una representación de los cuentos basada en una serie de acciones prototípicas (partida, villanía, rescate, etc) que se caracterizan por los personajes involucrados, el lugar donde acontecen y los objetos que posibilitan la acción. Como ejemplo, la Figura 5.7 muestra (a través de las capacidades de visualización de OntoBridge) la representación del cuento de Caperucita Roja. En la parte derecha de dicha figura se indica la complejidad (profundidad del árbol de instancias) y número de instancias de los distintos cuentos de la ontología.

El método de adaptación transformacional recibe tres listas: elementos a reemplazar, elementos a añadir y elementos a borrar. Dichos elementos pueden ser acciones, personajes, lugares u objetos. Estas tres listas constituyen la consulta del usuario y se utilizan para recuperar el cuento más similar a las restricciones indicadas en ellas (que existan o no ciertos elementos). A continuación se copia la solución del cuento más similar y se modifica de acuerdo con las listas anteriores. Para reemplazar elementos se utilizan las medidas de similitud expuestas en la Sección 5.4 para encontrar en la ontología las instancias más parecidas que pudieran servir de substitutas. Las listas de añadir y borrar se utilizan directamente insertando o eliminando

los elementos que contengan.

Mediante las transformaciones anteriores se consigue un nuevo cuento que cumple las restricciones que el usuario ha especificado en la consulta y que ha sido generado a partir de otro existente. Como ya vimos en la introducción de esta sección esta no es la única forma de adaptar una solución y también es posible generarlas reutilizando partes de las soluciones de los casos más similares. A continuación detallaremos el método de jCOLIBRI que sigue esta mecanismo denominado adaptación constructiva.

5.6.2 Adaptación Constructiva

Esta aproximación consiste en la reutilización de las soluciones de los casos más similares para construir una nueva solución para la consulta actual del sistema. En Plaza y Arcos (2002) la adaptación constructiva se describe como un proceso de búsqueda en el espacio de soluciones donde los casos se utilizan en dos fases: generación de hipótesis y ordenación de las hipótesis. De este modo, la adaptación constructiva puede interpretarse como una búsqueda heurística de primero el mejor por el espacio de soluciones que utiliza información de los casos ya resueltos para guiar dicha búsqueda. En el proceso de construcción de la solución se irá avanzando por diferentes estados de la solución hasta que esta alcance la calidad necesaria. Todo este proceso puede definirse algorítmicamente mediante el siguiente pseudocódigo:

Listado 5.1: Algoritmo de la Adaptación Constructiva según Plaza y Arcos (2002)

```
EA = crearLista(Estado-Inicial(Q));
Función CA(EA)
  Si esVacio(EA) entonces No-Solución;
  mejorEstado = primero(EA);
  Si Es-Objetivo(mejorEstado) entonces Sol(primero (EA));
  Si no
     ES = GH(mejorEstado);
     EA = HO(añadir(ES,quitar(EA, mejorEstado)));
     CA(EA);
```

En este código las funciones `GH` y `OH` son las de *Generación de Hipótesis* y *Ordenación de Hipótesis*. Las variables `EA` y `ES` son las listas de *Estados Abiertos* y *Estado Siguiente*. La función `Sol` convierte el estado solución en la solución del caso y finaliza la recursión. La función `Estado-Inicial` convierte los requisitos de entrada `Q` (la consulta) en un estado de búsqueda. Además, se utiliza la función `Es-Objetivo` para comprobar si el mejor estado actual es suficientemente bueno para ser la solución del caso.

jCOLIBRI incluye el algoritmo anterior ya implementado en un método. Dicho método únicamente necesita que se especifiquen las funciones `GH`. `OH`, `Estado-Inicial` y `Es-Objetivo`. Además hay que destacar que dicho algo-

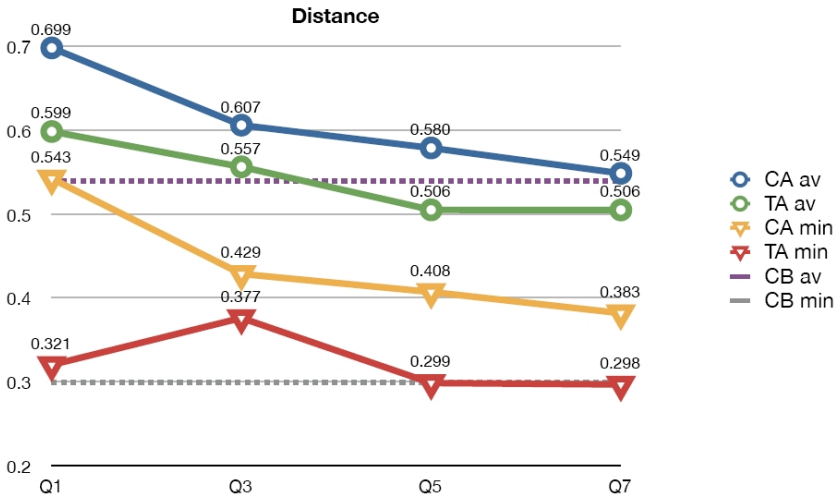


Figura 5.8: Comparación de adaptación constructiva y transformacional

ritmo no está restringido a la adaptación con ontologías sino que también es apropiado para adaptar cualquier otro tipo de casos.

Este método se ha utilizado también en el dominio de la generación de cuentos. En este caso, las soluciones se van generando progresivamente y cada una de ellas constituye un estado de búsqueda. La función de generación de hipótesis va añadiendo progresivamente nuevas acciones, lugares, personajes y objetos a cada solución parcial que se obtienen de los distintos cuentos incluidos en la base de casos. La función de ordenación de hipótesis utiliza una medida de similitud en ontologías (Sección 5.4) para medir cuanto se parecen los elementos que aparecen en cada solución parcial a los especificados por el usuario en la consulta. Finalmente, la solución es devuelta como correcta cuando ese valor de similitud supera un cierto umbral.

5.6.3 Comparación de las técnicas de adaptación

El principal objetivo de Díaz-Agudo et al. (2008b) consistía en compara los resultados obtenidos al aplicar adaptación transformacional y constructiva. La gran mayoría de sistemas CBR se basan en la suposición de que problemas similares tienen soluciones similares, sin embargo, en este trabajo se partía de la suposición inversa: un cuento generado mediante CBR no tiene que ser parecido a los existentes en la base de casos. Si se obtuvieran cuentos similares, el sistema no tendría éxito ya que lo que se busca en este tipo de tareas es la *originalidad*. Por lo tanto, al evaluar ambas aproximaciones se quería comprobar cómo se distribuían los cuentos generados por el espacio de soluciones posibles.

Para medir la distancia se utilizaron (de forma inversa) las medidas de

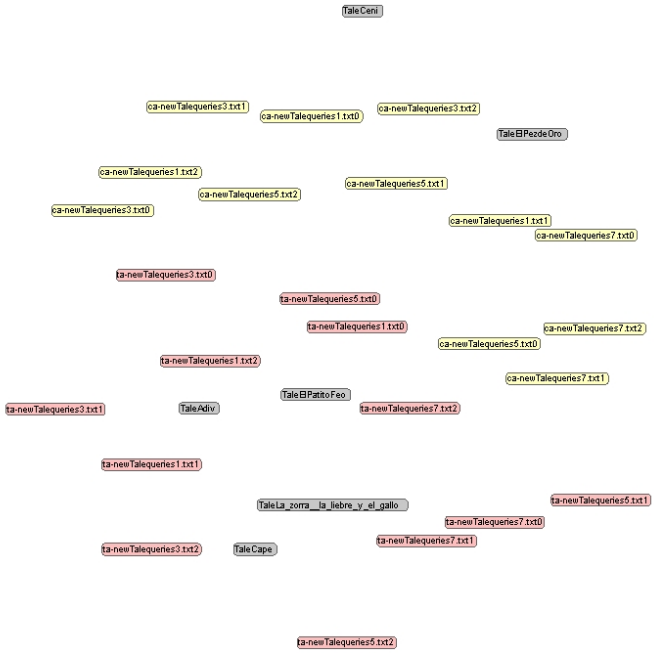


Figura 5.9: Distribución de cuentos generados mediante CBR

similitud con ontologías de jCOLIBRI. La distancia mínima entre cuentos de la base de casos era de 0.3 por lo que se tomó este valor como límite mínimo para los cuentos generados. Además, la distancia media entre los casos fue de 0.54 por lo que se tomó dicha medida como un valor de referencia al que los nuevos cuentos deberían acercarse. En el experimento se generaron cuatro grupos de consultas: $Q1$, $Q3$, $Q5$ y $Q7$ que se rellenaron respectivamente con 1,3,5 y 7 instancias de cada tipo (acciones, personajes, objetos y lugares). Cada grupo consistían en 20 consultas generadas aleatoriamente.

Los resultados obtenidos se muestran en la Figura 5.8. Como puede observarse, los valores en media (CA_{av} y TA_{av}) de ambas estrategias permanecen alrededor del valor medio de la base de casos (0.53) por lo que se concluye que las nuevos casos son igual de originales. Además nunca se baja del valor mínimo de 0.3 que se calculó a priori. Las gráficas también muestran que cuanto menos se restringe la consulta se obtienen resultados más originales ya que $Q1$ siempre obtiene mejores valores que $Q7$.

Otra forma de visualizar la originalidad de los casos generados es la mostrada en la Figura 5.9. Esta figura utiliza los métodos de visualización de la base de casos incluidos en jCOLIBRI para mostrar la distribución de los cuentos. Los cuentos en color gris son los originales de la base de casos, en color rosa aparecen los cuentos generados mediante adaptación transformacional, y en amarillo los generados por la adaptación constructiva.

Cómo última conclusión cabe destacar que antes de realizar los experimentos se pensaba que la adaptación transformacional obtendría resultados menos originales que la constructiva. Esta premisa se fundamentaba en que la aproximación transformacional se basa en uno de los cuentos que aparecen en la base de casos mientras que la constructiva genera los cuentos desde cero. Sin embargo, los resultados demostraron que ambas estrategias obtenían resultados más o menos similares.

5.7 Autoría de casos representados mediante ontologías

Es las secciones anteriores se han discutido las distintas aproximaciones para aprovechar la riqueza semántica de los casos representados por medio de ontologías. Aunque se han presentado varias maneras para acceder a una base de casos almacenada en la ontología y reutilizar las capacidades de razonamiento sobre ella para mejorar los procesos CBR, se ha dejado de lado el proceso de obtención de dicha representación.

Este problema es muy importante porque no sirve de nada desarrollar métodos de KI-CBR para ontologías si no se dispone de casos representados con ellas. Para superar este obstáculo, en Recio-García et al. (2006c) se presenta una aproximación basada en el Análisis Formal de Conceptos (FCA) para obtener casos representados mediante ontologías de forma semiautomática.

Este cuello de botella en la adquisición de los casos no es un problema exclusivo del KI-CBR sino que también se presenta en un campo de investigación mucho más amplio e indudablemente importante: la Web Semántica. En la Sección 2.2 ya se presentaron los objetivos de este área donde se pretende dotar de significado a las páginas web mediante ontologías. La investigación de la Web Semántica ha realizado muchos avances en las técnicas para manejar y dar utilidad a la información semántica, pero también sigue teniendo el problema de base de que sin documentos etiquetados mediante ontologías ninguna de las técnicas desarrolladas pudiere aplicarse.

Por lo tanto existe un claro paralelismo entre el problema de la creación de casos estructurados mediante ontologías y el problema de obtener documentos también anotados mediante ontologías. En el caso de la Web Semántica los documentos sobre los que se parte son páginas web que contienen texto libre, y en el CBR ya se ha comentado que la mayoría de las fuentes de conocimiento de casos suelen estar también representadas como textos. De este modo aparece como una idea bastante sensata la aplicación de las técnicas de CBR Textual para procesar la información contenida en los textos y obtener representaciones semánticas. Estas técnicas podrán ser luego aplicadas tanto a la anotación de documentos web en el campo Web Semántica como para la creación de casos estructurados mediante ontologías en el KI-CBR.

La técnica de anotación semántica que presentaremos a continuación utiliza tanto los métodos de CBR Textual incluidos en jCOLIBRI como otros métodos de Análisis Formal de Conceptos. El FCA es una técnica de análisis de datos que permite obtener dependencias entre los datos y que detallaremos en un apartado posterior. Antes, plantearemos el problema de la anotación semántica de documentos desde los dos puntos de vista: el de la Web Semántica y el del KI-CBR.

El gran sueño de Tim Berners-Lee de la Web Semántica puede resumirse en que los ordenadores sean capaces de entender la información presente en la Web. Sin embargo, parece que al menos en un futuro cercano las máquinas no serán capaces de entender dicha información directamente. Por lo tanto, las necesidades a más corto plazo se basan en información que sea procesable y comprensible para los ordenadores. Los pilares básicos para realizar esta tarea ya han sido establecidos Gil et al. (2005): lenguajes estandarizados para describir la semántica de los datos y programas que permitan el intercambio e interpretación de dicha información. Sin embargo, esta aproximación no tiene en cuenta una idea clave: ¿dónde y cómo se obtienen esos datos semánticos?

El proceso de obtener datos semánticos se ha calificado como *anotación semántica* Handschuh y Staab (2003), ya que normalmente consiste en marcar texto plano –que sólo es comprensible por personas– con metadatos semánticos representados en ontologías.

La anotación semántica es una tarea muy difícil y existen diferentes aproximaciones y herramientas que intentan llevar a cabo. Sin embargo, la gran mayoría de ellas se basan en anotaciones generadas por humanos. Estos procesos consisten normalmente en generar el contenido textual y el semántico de forma paralela pero siempre de forma manual. Estas aproximaciones tienen claros problemas de escalabilidad ya que el proceso de anotación involucra una importante cantidad de documentos y ontologías de gran envergadura que pueden estar incluso distribuidas. Estos inconvenientes son los que aparecen en la mayoría de herramientas de anotación manual[9] como SMORE, OntoMat Annotizer, COHSE, Ontomat and MnM.

También existen aproximaciones semiautomáticas basadas en técnicas de Extracción de Información (IE) para estructurar los documentos. Como ejemplo podemos citar KIM, Semantic World o Melita. El problema de esta aproximación es que el proceso de anotación requiere un amplio conjunto de conectores que permitan acceder a las distintas fuentes de información y que la extracción está normalmente limitada a catálogos de páginas con la misma estructura. Además, el mantenimiento de este tipo de aplicaciones es un problema complejo ya que es necesario adaptar los distintos conectores cada vez que cambia el formato del documento web Kushmerick et al. (1997). Finalmente, existe otro problema de completitud ya que es complicado co-

[9] http://annotation.semanticweb.org/tools/

nectar información que está relacionada de forma dispersa o de naturaleza subjetiva.

5.7.1 Anotación mediante métodos TCBR y Análisis Formal de Conceptos

Los métodos de CBR Textual encontrados en la literatura se centran normalmente en transformar datos textuales en casos semiestrucutrados que pueden ser manejados por los métodos CBR comunes. Este proceso es análogo a la anotación de los documentos de la Web Semántica ya que ambos procesos comparten el mismo objetivo: obtener una estructura que permita la indexación, recuperación y manipulación de los documentos/casos. Las aplicaciones de la Web Semántica utilizarán esta información estructurada para permitir a los agentes software que busquen y manipulen páginas web, mientras que la comunidad CBR utilizará estos datos en aplicaciones de ricas en conocimiento (KI-CBR).

Nuestra propuesta se basa en aplicar los métodos de CBR Textual presentados en el Capítulo 4 para procesar los documentos pero añadiendo un paso más donde se utilicen ontologías que definan la estructura semántica de los casos. En este proceso estamos reaprovechando dos recursos que nos ofrece el área de la Web Semántica y que ya hemos comentado anteriormente: un lenguaje estándar para la representación semántica de los datos como OWL, y repositorios de ontologías que puedan ser reutilizadas.

Para realizar el proceso de anotación proponemos la adaptación las reglas de extracción de información de los métodos de TCBR en jCOLIBRI respecto a una ontología concreta del dominio. Con esta transformación, los métodos de IE obtendrán fragmentos de información que corresponderán con conceptos de la ontología. Aún así, las reglas de extracción de información no son capaces de obtener las complejas estructuras de conceptos definidas en las ontologías. Para aclarar este problema volveremos al recomendador de restaurantes que sirvió para ilustrar los métodos de TCBR y que utilizaremos a lo largo de la explicación de nuestro método de anotación semántica. Obviamente, necesitaremos una ontología sobre el dominio que en nuestro caso obtuvimos adaptando varias ontologías ya existentes. En la ontología sobre restaurantes la descripción de un *restaurante* contiene una serie de *detalles de contacto*, *números de teléfono* y la *dirección*. Esta dirección está a su vez compuesta por la *calle*, su *tipo*, *nombre* y *número de puerta*. Cada uno de los nombres resaltados en cursiva son conceptos de la ontología pero sólo algunos de ellos (normalmente las hojas de dicha estructura) pueden ser extraídos mediante las reglas de IE. En este caso, los conceptos compuestos: restaurante, detalles de contacto, dirección y calle no serían inferidos por reglas simples de extracción de información.

Para resolver este problema de conexión de información dispersa hemos aplicado el Análisis Formal de Conceptos. Esta técnica nos permite completar la representación semántica de los casos ya que se utiliza para extraer dependencias entre los conceptos (etiquetas semánticas) que deberían aparecer pero que han sido omitidos por los métodos de IE. De este modo se mejora la riqueza semántica de los casos, lo que repercutirá en los posteriores procesos de indexación, recuperación y adaptación.

Nuestro método de anotación combina la anotación automática por medio de reglas de Extracción de Información y del Análisis Formal de Conceptos para obtener dependencias (reglas de asociación) entre las etiquetas, de modo que se ayude al experto humano en la tarea de anotación.

El FCA es una técnica matemática para el análisis de datos. Se describió por primera vez en Wille (1982) y desde entonces ha sido aplicado en distintas áreas (Wolff, 1993). FCA distingue entre *objetos formales* (o entidades) y *atributos formales* (o características). Nosotros consideramos cada texto (o caso) etiquetado manualmente como un objeto, y cada posible etiqueta semántica como los atributos. El algoritmo del FCA recibe una relación binaria llamada *contexto formal* que relaciona los objetos y los atributos formales. El contexto suele representarse como una tabla de incidencia donde las filas representan a los objetos y las columnas representan los atributos. Las celdas de esta tabla aparecerán marcadas si el objeto de dicha fila presenta el atributo correspondiente a la columna.

En nuestro caso, consideramos los casos (o páginas web) como objetos formales y las etiquetas de la ontología que podrían utilizarse en su anotación como los atributos formales. El contexto formal generado tendrá tantos objetos como textos y tantos atributos como etiquetas de la ontología. Cada texto estará relacionado con las etiquetas si éstas aparecen en su anotación generada manualmente. La Figura 5.10 muestra la descripción textual de un restaurante (a), su versión anotada con la ontología (b) y una parte del contexto formal donde aparece dicho restaurante.

Mediante el contexto formal, FCA es capaz de construir un conjunto de *conceptos formales* (o simplemente conceptos). Formalmente hablando, un concepto es un par (A, B) donde A es el conjunto de objetos (conocido como *extensión*) y B es el conjunto de atributos comunes de esos objetos (*intensión*). Los conceptos de un determinado contexto pueden ser ordenados utilizando una relación de subconcepto-superconcepto que puede ser representada como un *retículo*, tal y como se muestra en la Figura 5.13(a).

Aunque los conceptos formales y la estructura del retículo podrían ser útiles por sí mismos (Díaz-Agudo y González-Calero, 2001a), nosotros utilizaremos la posibilidad de extraer reglas de dependencia a partir ellos. Una regla de dependencia es una expresión $A \rightarrow B$ donde A y B son conjuntos de atributos. Esta regla significa que cualquier objeto que tenga todos los

Alegria
3510 Sunset Blvd.
Silver Lake
(323) 913-1422
The best food here revolves around ...

(a) Descripción textual del restaurante

```
...
<restaurant:contact>
<contact:ContactDetails>
<contact:phoneNumber
   rdf:datatype="string">
(323) 913-1442
</contact:phoneNumber>
<contact:address>
<address:Address>
<address:areaName
   rdf:datatype="string">
Silver Lake
</address:areaName>
<address:addressPublicPlace>
<address:Boulevard
   rdf:ID="Sunset_Blvd."/>
</address:addressPublicPlace>
<address:cityName
   rdf:datatype="string">
Silver Lake
</address:cityName>
<address:doorNumber
   rdf:datatype="string">
3510
</address:doorNumber>
</address:Address>
</contact:address>
</contact:ContactDetails>
</restaurant:contact>
...
```

(b) Restaurante etiquetado

	contact	contactDetails	phoneNumber	addressPublicPlace	Boulevard	cityName	doorNumber	price	startHour	TakeAwayFacility	atmospheres	...
Alegria	■	■	■	■	■	■	■	■	■	■		...
Alex	■	■	■	■		■	■	■			■	...
Antique	■	■	■	■		■	■				■	...
Beverly	■	■	■			■						...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

(c) Contexto Formal

Figura 5.10: Ejemplo de FCA

atributos de A tendrá probablemente los atributos en B.

Las reglas de dependencia se caracterizan por dos parámetros: *confianza* (confidence) y *respaldo* (support). La confianza expresa la probabilidad de que se cumpla la regla, en otras palabras, es el porcentaje de objetos que teniendo todos los atributos de A también tienen los de B. Por otro lado, el respaldo indica el número de objetos donde la regla es aplicable, o lo que es lo mismo, el número de objetos con atributos en A y B dividido por el número total de objetos.

Los algoritmos de extracción de reglas basados en el FCA son capaces de obtener eficientemente todas las reglas de dependencia que superan un grado de confianza determinado. Aunque existen varios algoritmos nosotros utilizamos los de Guigues y Duquenne (1986) y Luxenburguer (1991).

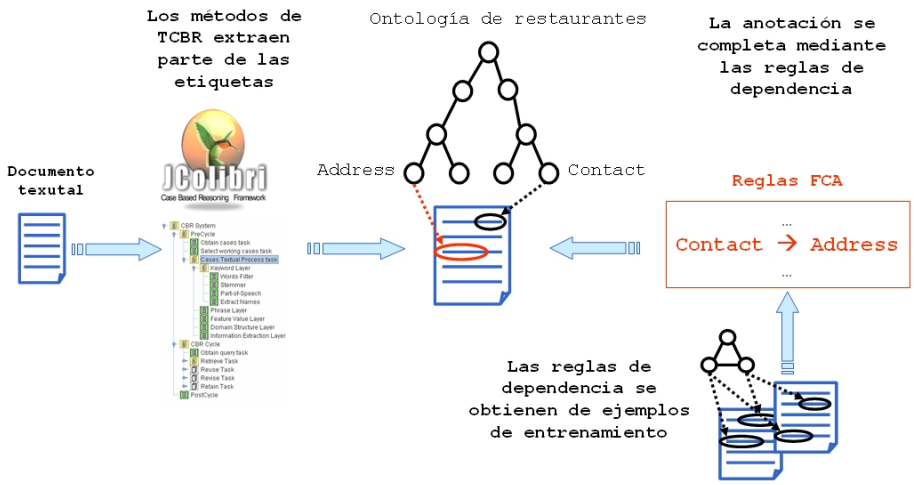


Figura 5.11: Proceso de anotación semántica de casos

Nuestro método de anotación comienza con un conjunto de casos textuales (C_1) que han sido anotados manualmente. Llamaremos $L_M(C_1)$ al conjunto de etiquetas utilizadas para anotar los casos de C_1. A continuación, aplicamos FCA para extraer las regla de dependencia entre los atributos (o etiquetas). Este conjunto de reglas, $R = \text{fca}(L_M(C_1))$, se utilizará más adelante en el proceso de adaptación.

Por ejemplo, R podría incluir una regla como:

`restaurant:contact -> address:Address`

ya que todos los textos de C_1 que tienen la etiqueta que representa los datos de contacto (`restaurant:contact`) también contienen la etiqueta para definir la dirección (`address:Address`) (quizás con cierto grado de confianza).

Cuando nuestro método recibe un nuevo texto T para ser anotado, se aplican primero las reglas de IE para obtener una primera versión de sus etiquetas, $L_{IE}(T)$. Como ya se comentó anteriormente, las reglas de IE no extraerán todas las etiquetas que deberían ser asociadas al caso debido a las limitaciones de la representación estructurada mediante árboles de conceptos. Para mejorar esos resultados, el método aplica las reglas de dependencia R para descubrir las etiquetas que no fueron inferidas por las reglas de IE. De este modo se obtiene el conjunto final de etiquetas $L_{FCA} = Apply(R, L_{IE}(T))$.

Continuando el ejemplo, si $L_{IE}(T)$ contiene la etiqueta de la dirección pero carece de la etiqueta sobre los datos de contacto, la aplicación de R al conjunto de etiquetas obtenido mediante IE devolverá esta última etiqueta. Todo el esquema completo del método de adaptación se muestra en la Figura 5.11 donde pueden observarse los distintos elementos involucrados en la anotación semántica de un texto.

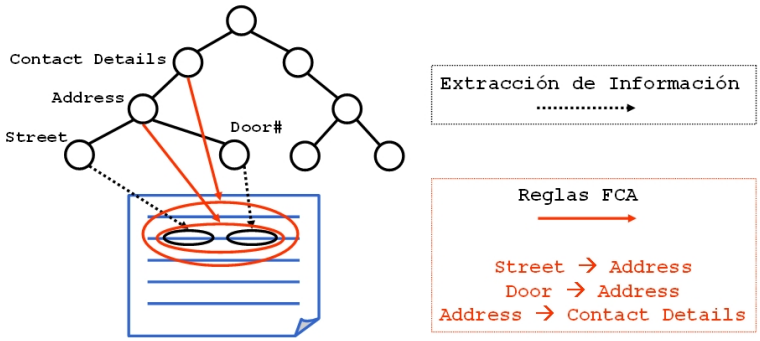


Figura 5.12: Anotación completa de la estructura del caso

Si se aplican en cadena las distintas reglas FCA, nuestro método permite indicar al experto humano encargado de la anotación qué etiquetas de la estructura interna del árbol de conceptos no han sido extraídas por las reglas de IE pero deben ser añadidas a la representación. Dependiendo de los grados de confianza y respaldo configurados, el anotador humano dispondrá de las etiquetas que probabilísticamente deberían aparecer en el documento para completar la representación tal y como se muestra en la Figura 5.12

Para probar la mejora en el proceso de adaptación a través de nuestro método, se ejecutaron una serie de experimentos sobre textos con descripciones de restaurantes. En esos experimentos se comparan el conjunto de etiquetas obtenidas manualmente (L_M) con las etiquetas devueltas por las reglas de IE (L_{IE}) y el conjunto final de etiquetas una vez aplicadas las reglas de dependencia (L_{FCA}). A continuación detallaremos dichos experimentos.

5.7.2 Resultados experimentales

Para poder experimentar con el método de anotación, se necesitaron dos elementos básicos: una ontología y un conjunto de textos sobre el dominio de los restaurantes.

La ontología se generó a partir de otras ontologías utilizadas en el proyecto *Agentcities* [10]. Estos recursos se encontraban originalmente en el lenguaje DAML+OWL (Connolly et al., 2001) por lo que hubo que traducirlos al lenguaje OWL. La ontología generada combina varias subontologías (sobre direcciones, precios, fechas y comida) y contiene más de 1000 conceptos aunque sólo unos cuantos son utilizados en el proceso de etiquetado. Esta ontología está publicada para el acceso libre en *http://gaia.fdi.ucm.es/ontologies/restaurants.owl*.

[10] http://www.agentcities.org

La base de casos consta de 268 documentos textuales con información sobre restaurantes que fueron obtenidos de un sitio web real[11].

El objetivo del experimento es comparar el método de anotación semi-automático con el proceso manual. Para ello, se anotaron manualmente un conjunto de textos con las etiquetas de la ontología. Este proceso manual es realmente complejo y tedioso así que solamente se utilizó un conjunto de entrenamiento con 30 restaurantes. Por otro lado, el desarrollo de las reglas de IE adaptadas a la ontología costó como cuatro veces menos tiempo que el proceso manual. De este modo, si nuestro método de anotación obtuviera resultados similares a la anotación manual, la mejora de todo el proceso sería muy notable. En lo que resta de sección nos referiremos a este conjunto de 30 textos como C y a su versión anotada manualmente como $L_M(C)$.

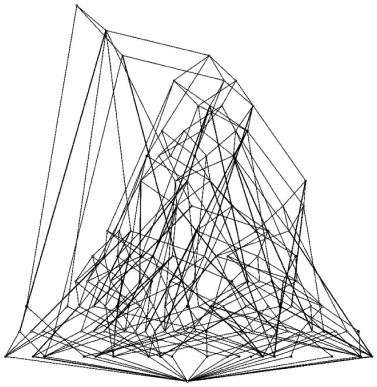
Para respaldar los resultados, el proceso se ha duplicado aplicando el método de anotación en dos experimentos A y B. En ambos, se ha dividido el conjunto de restaurantes C en dos grupos diferentes, $C_{1\{A,B\}}$ y $C_{2\{A,B\}}$, con 20 y 10 restaurantes respectivamente. Entonces se ha aplicado FCA a C_1 para extraer sus reglas de dependencias. Dichas reglas se aplican para completar el etiquetado mediante IE sobre C_2. Finalmente se compara el conjunto de etiquetas final ($L_{FCA}(C_2)$) con las etiquetas de C_2 obtenidas manualmente ($L_M(C_2)$).

Los conjuntos de entrenamiento A y B se eligieron intencionadamente para reflejar el mejor y el peor escenario. El conjunto C contiene varias descripciones irregulares que no presentan la misma información que las otras porque algunos datos –y por consiguiente sus etiquetas– se omiten. Los experimentos A y B contienen esas descripciones en C_1 y C_2. De este modo, el experimento A contiene las descripciones irregulares en el conjunto al que se aplica FCA: C_{1A}. Por otro lado, el experimento B tiene esas irregularidades en el conjunto de restaurantes anotados manualmente cuyo etiquetado se extiende con las reglas FCA: $C2B$. Con esta partición se pretende comprobar cómo afectan los ejemplos irregulares a la eficiencia del método.

Cada experimento se divide en cuatro etapas:

1. Consiste en analizar con FCA ambos conjuntos de entrenamiento C_{1A} y C_{1B}. Como se explicó anteriormente, las reglas de dependencia pueden extraerse para que cumplan un grado de confianza mínimo. Por ello, en vez de utilizar el 100 % (reglas exactas) se utilizaron diversos grados que permitieran inferir cómo afecta este parámetro a los resultados finales. Concretamente, se extraen los conjuntos de reglas que varían desde el 100 % (R_{A100} y R_{B100}) al 80 % (R_{A80} y R_{B80}) de confianza en decrementos del 5 %. Para ilustrar la complejidad de la base

[11] http://www.laweekly.com/eat-drink



(a) Retículo del conjunto de entrenamiento A

Confianza	A	B
100 %	137	117
95 %	138	121
90 %	168	166
85 %	183	176
80 %	192	180

(b) Número de reglas de dependencia

Figura 5.13: Resultados del FCA

de casos, la Figura 5.13 muestra el retículo asociado al conjunto de entrenamiento A ($C1A$) que tiene 135 conceptos formales. Además se muestra el número de reglas de dependencia obtenidas en cada caso.

2. En segundo lugar se aplican las reglas de IE a la versión en texto plano de los 10 restaurantes restantes. Estas reglas son una adaptación de las utilizadas en el experimento explicado en la Sección 4.4.1 (Recio-García et al., 2005a). Mediante los métodos de CBR Textual de jCOLIBRI se obtiene el conjunto etiquetas $L_{IE}(C_{2A})$ para el experimento A, y respectivamente $L_{IE}(C_{2B})$ para B. Denominaremos $L_{IE}(C_2)$ al conjunto de etiquetas obtenidas al aplicar IE a los textos planos. Como se expondrá en el próximo apartado, el número de etiquetas en $L_{IE}(C_2)$ es aproximadamente el 40 % del total de etiquetas que se obtienen mediante la anotación manual $L_M(C_2)$.

3. Este tercer paso consiste en aplicar las reglas FCA obtenidas en el paso 1 a los restaurantes anotados mediante IE. Obviamente, se utiliza R_{Ax} para mejorar el etiquetado $L_{IE}(C_{2A})$ y R_{Ax} para $L_{IE}(C_{2B})$.

4. En el último paso se compara el etiquetado propuesto por las reglas FCA contra las etiquetas en M_1.

Aunque sólo tenga un valor teórico, también se realiza un experimento extra para compararlo con los otros dos. Este experimento se referenciará en las próximas líneas como el "Conjunto Completo". Aquí se han extraído las reglas de dependencia a todo el conjunto de restaurantes anotados (C). Entonces se aplica IE al mismo conjunto de restaurantes y se utilizan las reglas

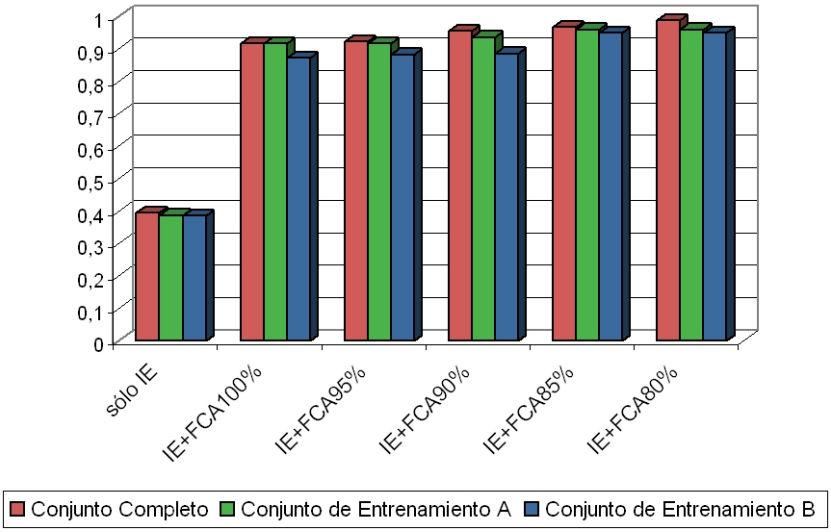


Figura 5.14: Exhaustividad del método de anotación semántica

de dependencia para completar la anotación. Visto de otro modo, se está aplicando el método de anotación sobre el mismo conjunto de documentos que se utilizó para entrenarlo. Este experimento no tiene mucha relevancia práctica pero sí que nos indica el límite máximo de exhaustividad del método. Si el proceso fuera perfecto este experimento devolvería una precisión y exhaustividad perfecta ya que se comprueban los resultados con el mismo conjunto utilizado en el entrenamiento.

Para medir la eficiencia de los múltiples experimentos realizados se han vuelto a utilizar las medidas de *precisión* y *exhaustividad*. A continuación presentaremos los resultados obtenidos al comparar la anotación manual con la anotación mediante IE para finalizar cotejando la anotación manual y la anotación con IE+FCA.

Al realizar la comparación entre L_M y L_IE hay que tener en cuenta que las reglas de IE sólo son capaces de extraer las etiquetas de las hojas de la estructura de conceptos. La razón es que los nodos internos de dicho árbol representan elementos abstractos que no pueden ser obtenidos directamente del texto. Este problema de conexión de información dispersa ya se detalló en la Sección 5.7.1.

Consecuentemente, si la descripción semántica de un restaurante utiliza unos 40 conceptos se generan reglas de IE para 20 de ellos. Con este número de reglas el proceso de extracción de información debería ser capaz de obtener el 50 % de las etiquetas necesarias en la anotación completa de un

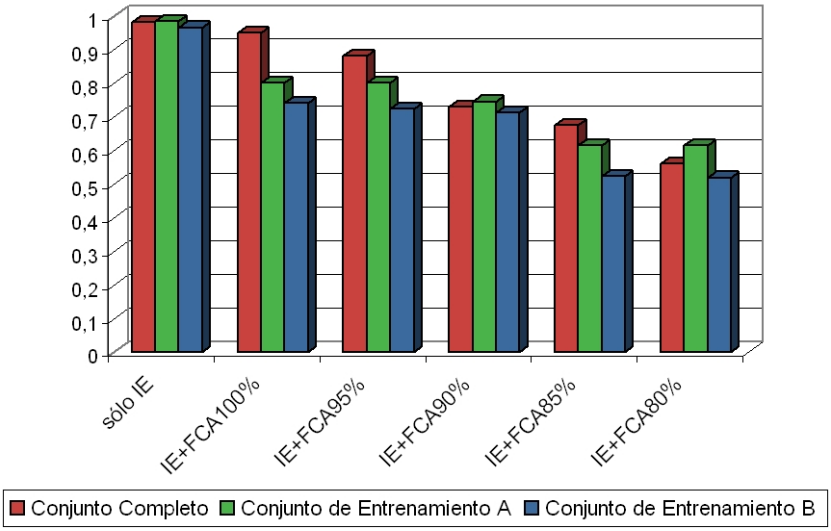


Figura 5.15: Precisión del método de anotación semántica

restaurante. Los resultados experimentales muestran que ese valor baja hasta el 40 % de media. Este valor se representa en el primer grupo de columnas (*sólo IE*) de la Figura 5.14 donde se muestran los valores de exhaustividad obtenidos en los experimentos. Este valor podría interpretarse como un valor bajo ya que los otros experimentos obtienen mejores valores, pero esto está justificado en que nuestro objetivo no consistía en mostrar las posibilidades de las reglas de IE sino en desarrollar sus reglas rápidamente y completarlas con las de FCA para mejorar el tiempo y esfuerzo invertidos en todo el proceso. Respecto a la precisión obtenida, ésta se mantuvo por encima del 98 % del total de etiquetas según se muestra en el primer grupo de columnas de la Figura 5.15. El 2 % restante son los denominados "falsos positivos" devueltos por los métodos de IE.

El resultado más importante de los experimentos es la comparación entre el etiquetado manual L_M y el etiquetado mediante las reglas de IE+FCA L_{FCA}. Estos resultados aparecen en la Figura 5.14 y muestran que la exhaustividad aumenta del 40 % al 90 %. En esta figura se comprueba que la exhaustividad aumenta según se desciende el grado de confianza de las reglas. Cada grupo de columnas representa el mismo experimento pero con diferentes grados de confianza (IE+FCA100 %, IE+FCA90 %, ...).

De forma opuesta, si se aumenta la confianza se obtiene una precisión más baja. Obviamente, esto es un efecto directo de la confianza porque cuanto más genéricas sean las reglas serán más propensas a generar "falsos positivos". Como se muestra en la Figura 5.15, una confianza inferior al 90 % reduce de-

masiado la precisión. Por lo tanto, los mejores resultados de nuestro método se obtienen con una confianza superior al 90 %.

En ambas figuras (5.14 y 5.15) la columna que representa el "Conjunto completo" muestra los resultados al utilizar todos los ejemplos de entrenamiento. Por su parte, las columnas "Conjunto de Entrenamiento A" y "Conjunto de Entrenamiento B" representan los valores resultantes de utilizar los conjuntos de entrenamiento C_{1A} y C_{1B}.

Tal y como se explicó en el apartado anterior, el "Conjunto Completo" es un indicador teórico de la eficiencia del método y los resultados muestran que ese valor teórico es siempre mayor que el obtenido experimentalmente. Es importante remarcar que los valores de exhaustividad son bastante parejos al utilizar distintos conjuntos de entrenamiento mientras que la precisión varía considerablemente. Este resultado significa que la principal ventaja del método de anotación es que recupera casi todas las etiquetas que aparecían en los ejemplos de entrenamiento. Contrariamente, la principal desventaja es que se recuperan bastantes etiquetas incorrectas. Este problema es especialmente significativo cuando se utiliza un nivel de confidencia en las reglas menor del 90 %.

Gracias a la duplicidad de los experimentos A y B es posible extraer otra conclusión. El experimento A contiene ejemplos irregulares en el conjunto utilizado para extraer las reglas FCA (C_{1A}) mientras que el experimento B tiene esos ejemplos en el conjunto al que se mejora el etiquetado mediante las reglas FCA C_{2B}. Como los resultados de A son mejores que los de B es posible concluir que la generación de las reglas FCA oculta los errores producidos por las descripciones irregulares. El experimento B presenta peores resultados porque sus reglas FCA son similiares a las de A y es en el conjunto C_{2B} donde se encuentran los ejemplos defectuosos. Aunque estos ejemplos se mejoren con las reglas de dependencia, los resultados obtenidos siguen siendo peores.

La conclusión global del experimento es que las reglas de FCA mejoran considerablemente la eficiencia del proceso de extracción de información. En teoría, nuestro escenario restringe el rendimiento de la IE al 50 %. En la práctica se obtiene un 40 % pero se gana en la agilidad de la generación de las reglas de IE ya que no se invierte demasiado tiempo en ellas. Completando ese etiquetado con las reglas FCA se incrementa automáticamente este valor al 90 % (perdiendo únicamente el 15 % de la precisión).

El aumento en el valor de la exhaustividad indica que el sistema extrae casi todos los conceptos en la ontología que no pueden ser obtenidos mediante Extracción de Información. De este modo, el sistema podría proponer automáticamente conceptos de la ontología durante el proceso de anotación semántica de los casos.

Como resumen final, nuestro método de anotación semántica permite generar casos representados mediante ontologías a partir de descripciones tex-

tuales. Esta representación permite aplicar a los distintos métodos de recuperación y adaptación de KI-CBR presentados en este capítulo.

Para desarrollar este método semiautomático se ha reaprovechado el paralelismo existente entre el CBR y el campo de la Web Semántica. En ambas áreas se necesita obtener una representación semántica por medio de ontologías que permita manejar los documentos textuales que contienen el conocimiento. De esta forma se propone la reutilización de los recursos que la Web Semántica puede aportar al CBR: estándares de marcado como OWL y ontologías ya generadas que permiten reutilizar su información.

5.8 Conclusiones

La principal fuente de conocimiento en un sistema CBR son los casos. Sin embargo, también es necesario incorporar otro tipo de conocimiento complementario que se utilizará principalmente en las etapas de recuperación y adaptación. Este conocimiento adicional es especialmente necesario en las aplicaciones donde existen pocos casos o donde estos casos no aportan toda la información necesaria para realizar los razonamientos. Este tipo de aplicaciones que utilizan una gran cantidad de conocimiento complementario al de los casos se denominan sistemas CBR con conocimiento intensivo o KI-CBR.

La gran mayoría de sistemas CBR incluye este conocimiento adicional de forma implícita en los distintos métodos que forman el ciclo CBR. Sin embargo, este acoplamiento con la información sobre el dominio no permite su reutilización en otros escenarios. Como el objetivo principal de jCOLIBRI es ofrecer métodos reutilizables, es necesario desacoplar los métodos del conocimiento del dominio de modo que se evite el típico problema del cuello de botella presente en los sistemas basados en conocimiento. Una forma muy viable de separar métodos CBR y conocimiento sobre el dominio es representar este último de forma explícita en ontologías. Como ya se comentó al describir las ontologías, este tipo de representación de la información es especialmente útil para encapsular el conocimiento sobre el dominio de una aplicación CBR e incorporarlo al sistema. El uso de ontologías permite su reutilización en diversas aplicaciones y ofrece la posibilidad de utilizar en los procesos CBR sus posibilidades de inferencia y razonamiento.

La primera aportación de las ontologías al CBR discutida en este capítulo consiste en la posibilidad de utilizarlas como medio de persistencia de los casos. De este modo, la estructura de conceptos de la ontología sirve para organizar las instancias que forman los casos, es decir, las instancias son los distintos valores que puede tomar un atributo.

En jCOLIBRI la persistencia en ontologías presenta dos variantes:

- Utilizar las instancias de la ontología como los posibles valores de algún

atributo del caso. Esto permite almacenar los casos en cualquier otro medio de persistencia (como B.B.D.D. o ficheros de textos) y enlazar alguno de esos atributos con instancias. Para ello, la representación del atributo en el medio de persistencia deberá contener el identificador de la instancia en la ontología. Dicho atributo estará tipado mediante la clase `Concept` que permite enlazar automáticamente con la instancia correspondiente a través de OntoBridge.

Mediante este esquema de mapeo se permite aplicar medidas de similitud basadas en ontologías y mecanismos de adaptación que también explotan este tipo de representación.

- La otra opción consiste en mapear completamente la estructura del caso en la ontología. Para ello, la ontología enlazará las distintas instancias que forman el caso a través de propiedades y cada instancia será mapeada a alguno de los atributos del caso que maneja jCOLIBRI. Esta aproximación tiene la ventaja de que no es necesario ningún otro mecanismo de persistencia auxiliar.

La utilización de instancias como valores para los atributos del caso permite aplicar medidas de similitud que aprovechan las características de razonamiento de las ontologías. Las medidas implementadas en nuestro armazón se basan en la posición de los distintos elementos dentro de la jerarquía conceptual de la ontología. Estas medidas se clasifican en medidas de similitud por contenidos (*intra-class*) y medidas de similitud por posición (*inter-class*). El primer tipo se basa en la similitud por posición para comparar dos arboles de instancias (es decir, la estructura resultante de la unión de instancias mediante propiedades). El segundo tipo de medidas por posición simplemente depende de la posición de las instancias en la taxonomía. jCOLIBRI incorpora la función *intra-class* y cuatro funciones inter-class: *similitud por profundidad*, *profundidad*, *básica*, *coseno* y *detalle*. Estas medidas son utilizadas por el algoritmo k-NN del armazón como funciones de similitud locales que comparan dos atributos.

La inclusión de valores procedentes de ontologías en los atributos del caso implica una mejora considerable en la expresividad de los mismos y de las posibles consultas al sistema CBR.

Respecto al *vocabulario del caso* existe una aproximación directa basada en el paradigma de orientación a objetos: los conceptos son los tipos y las instancias son los valores permitidos para dichos tipos. Esta es la aproximación más común aunque no tiene que ser la única. Por ejemplo, se podría permitir que un atributo del caso pudiera tomar como valor cualquier instancia de la ontología.

Esta relajación en las restricciones de valores (instancias) que puede tomar un atributo es especialmente útil a la hora de definir la consulta. Esta opción permite utilizar consultas más expresivas si se utilizan las diferentes instancias de la ontología además de las instancias que aparecen en los atributos del caso. De esta forma se permite que el usuario defina sus preferencias mediante cualquier instancia de la ontología y a continuación se utilizan las medidas de similitud con ontologías para calcular su similitud con los diferentes casos.

La última gran aportación de las ontologías al ciclo CBR la encontramos en la etapa de adaptación. Esta etapa consiste en modificar la solución de los casos respecto a las restricciones de la consulta. Esta tarea necesita mucho conocimiento sobre el dominio por lo que muchos sistemas CBR lo incluyen de forma implícita. Sin embargo, al utilizar ontologías podemos reutilizar el conocimiento sobre el dominio necesario para adaptar las soluciones.

En jCOLIBRI se incluyen dos métodos de adaptación: transformacional y constructiva. La *adaptación transformacional* recupera la solución del caso más similar y realiza sucesivas modificaciones sobre ella para adaptarla a la consulta. En nuestro método dichas transformaciones se realizan a través de una serie de instrucciones que indican qué elementos de la solución hay que añadir, eliminar o substituir. Estas instrucciones pueden expresarse a través de reglas o simplemente como listas de instancias que se incluirán o borrarán directamente y que se substituirán por otras que sean las más parecidas según las medidas de similitud basadas en ontologías que incorpora el armazón.

El otro mecanismo de adaptación es la adaptación constructiva. En este caso se crea una nueva solución a partir de las soluciones de los casos más similares. En jCOLIBRI se sigue el algoritmo de Plaza y Arcos (2002) donde este proceso se plantea como una búsqueda heurística por el espacio de posibles soluciones. Para ello, cada solución parcial que esté siendo generada constituirá un estado de búsqueda que será generado y ordenado por sendas funciones dependientes de la aplicación. Cuando una de las soluciones alcance una calidad determinada será devuelta como definitiva.

Los métodos descritos anteriormente proponen distintas aproximaciones para aprovechar la riqueza semántica de los casos representados por medio de ontologías. Aunque se han presentado varias maneras para acceder a una base de casos almacenada en la ontología y reutilizar las capacidades de razonamiento sobre ella para mejorar los procesos CBR, se ha dejado de lado el proceso de obtención de dicha representación.

Este problema es muy importante porque no sirve de nada desarrollar métodos de KI-CBR para ontologías si no se dispone de casos representados con ellas. Este cuello de botella en la adquisición de los casos no es un problema exclusivo del KI-CBR sino que también se presenta en un cam-

po de investigación mucho más amplio como el de la Web Semántica. La investigación en este área pretende dotar de significado a las páginas web mediante ontologías y, aunque ha realizado muchos avances en las técnicas para manejar la información semántica, sigue teniendo el problema de que sin documentos etiquetados mediante ontologías ninguna de las técnicas desarrolladas es aplicable.

Para superar este obstáculo, se presenta una aproximación basada en el los métodos de CBR Textual de jCOLIBRI y el Análisis Formal de Conceptos (FCA) para obtener casos representados mediante ontologías de forma semiautomática. El proceso de anotación propuesto adapta las reglas de extracción de información de los métodos de TCBR respecto a una ontología concreta del dominio. Esta extracción de información nunca será completa y no obtendrá todas las etiquetas necesarias para describir el caso. Para solucionarlo se utiliza el FCA generando una serie de reglas que permiten completar el conjunto de etiquetas necesario.

Por último cabe destacar que todos los métodos necesarios manejan la información contenida en las ontologías mediante una librería generada ad-hoc denominada OntoBridge. Esta librería oculta la complejidad de la gestión de ontologías y la conexión a los razonadores de Lógicas Descriptivas que realizan la inferencia.

La experiencia en el desarrollo de OntoBridge y en el marcado semiautomático de documentos ha hecho posible desarrollar una metodología educativa para la enseñanza de los conceptos básicos de la Web Semántica y el marcado de documentos. Esta metodología se presenta en Díaz-Agudo et al. (2008a).

El trabajo descrito en este capítulo describe un conjunto de métodos que suponen una aportación al desarrollo de aplicaciones KI-CBR ya que desacoplan el conocimiento complementario de los casos por medio de ontologías. Dichos métodos permiten utilizar las ontologías como medio de persistencia de los casos y para mejorar el proceso de recuperación y adaptación. Además se presenta un mecanismo para generar semiautomáticamente casos anotados con ontologías a partir de documentos textuales.

Capítulo 6

Diseño de sistemas CBR basado en plantillas

– Por fin tengo clavos suficientes y una buena caja donde guardarlos... ¡a por el martillo!

Este capítulo presenta los mecanismos incluidos en jCOLIBRI 2 para asistir a los diseñadores en el proceso de diseño de sistemas CBR con jCOLIBRI. La aproximación elegida consiste en abstraer el comportamiento de los sistemas CBR en un conjunto de plantillas que puedan ser reutilizadas según los requisitos de la aplicación a diseñar. En la Sección 3.1.1 se presentó el proceso de diseño de sistemas en jCOLIBRI 1 que se basaba en las ideas de UPML. En este proceso se realiza una descomposición en tareas que define el comportamiento de la aplicación y a continuación se asigna un método a cada una de estas tareas que incluye la funcionalidad necesaria para resolverlas. Sin embargo, este proceso presenta algunos inconvenientes entre los que destaca la limitación de descomponer las aplicaciones en secuencias lineales de tareas. Para solventar este problema, en jCOLIBRI 2 proponemos un proceso de diseño de sistemas CBR basado en plantillas. Cada plantilla abstrae varios sistemas CBR y puede definirse mediante flujos de control más complejos (p.e. condiciones o bucles). Estas plantillas seguirán estando formadas por tareas a las que se asignarán métodos que las resuelvan.

El proceso de diseño de sistemas CBR que se presenta en este capítulo sigue también una aproximación CBR: la librería de plantillas representa experiencias de diseño de sistemas CBR y forma una base de casos a partir de la cual el usuario recupera la más apropiada para la aplicación que desea implementar. Una vez recuperada dicha plantilla, el usuario modifica los métodos que resuelven las tareas para adaptar el sistema a sus requisitos. De esta forma, el proceso de diseño de sistemas CBR en jCOLIBRI es a su

vez un proceso CBR. Esta aproximación se aprovecha de la experiencia en construcción de sistemas CBR acumulada durante el desarrollo del armazón. Por lo tanto, las herramientas aquí presentadas aplican varios de los procesos de recuperación y adaptación desarrollados durante el trabajo de tesis. Estas dos etapas de recuperación y adaptación de plantillas se presentarán en las Secciones 6.4 y 6.5 respectivamente.

Desde el punto de vista de la arquitectura de jCOLIBRI 2, las herramientas planteadas en este capítulo se enmarcan en la capa superior orientada a diseñadores de la Figura 3.11 (pág. 94) que se detalla en el Capítulo 3. Como se explica en ese capítulo, esta parte de la arquitectura se fundamenta en las tecnologías surgidas en la Web Semántica y los Servicios Web Semánticos (presentados en la Sección 2.3.2 del estado del arte) para ofrecer una serie de herramientas que facilitan la composición de los métodos del armazón. Gracias a estas tecnologías, en este capítulo se plantea una representación semántica de las plantillas que permite razonar sobre su estructura y funcionamiento para así automatizar (al menos en parte) el proceso de selección y asignación de métodos.

En la Figura 3.11 pueden observarse los componentes principales de la arquitectura de esta capa orientada a diseñadores, donde aparece la librería de plantillas, los métodos que han sido presentados en los capítulos anteriores de esta memoria, su descripción semántica y el papel central de CBROnto como eje principal de todo el mecanismo de composición.

La siguiente sección describe el proceso de diseño de sistemas CBR basado en plantillas, para después describir la organización de la librería de plantillas y el dominio de aplicación elegido: los sistemas de recomendación. Finalmente, las secciones 6.4 y 6.5 detallan las etapas de recuperación y adaptación de plantillas.

6.1 Proceso de diseño de sistemas CBR mediante plantillas

El proceso de diseño de sistemas CBR se basa en la interacción con las herramientas de jCOLIBRI 2 para configurar el sistema deseado. Como ya se comentó en la introducción, este proceso sigue un esquema CBR: utilizar una librería de plantillas como base de casos, recuperar la más parecida al sistema a implementar y adaptarla a las necesidades específicas del diseñador.

Esta aproximación permite el desarrollo rápido de prototipos CBR y además es muy adecuado desde el punto de vista académico, tal y como se presentará en el Capítulo 7. Terminológicamente, nos referiremos a este proceso de generación de sistemas CBR como *diseño basado en plantillas*.

En contraposición, denominaremos *diseño basado en métodos* al desarrollo de aplicaciones CBR mediante la utilización directa del código de la parte de caja blanca de jCOLIBRI 2.

La eficiencia e idoneidad del diseño basado en plantillas depende en gran medida de que las plantillas sean desarrolladas por expertos que tengan experiencia suficiente en construir sistemas CBR de un determinado tipo y por lo tanto sean capaces de abstraer el comportamiento de las aplicaciones que hayan construido. Este tipo de diseño presenta además una serie de retos de investigación que han sido resueltos durante el trabajo de tesis:

- La primera cuestión a resolver es la adquisición de un conjunto de plantillas que sea suficientemente representativo. Tener una base de casos de plantillas requiere una discusión sobre la granularidad, genericidad, diversidad y cobertura de la base de casos. Además es necesario representar las dependencias entre los elementos de las plantillas, y anotarlas junto a sus componentes. Por último cada plantilla podrá tener distintos puntos de variabilidad, por ejemplo, tareas que puedan ser resueltas de diversas maneras utilizando métodos distintos.

  Las Sección 6.2 detalla la forma de resolver la cuestión inicial de la obtención de la librería de plantillas, mientras que la Sección 6.3.1 se encarga de explicar su representación. Ambos aspectos del desarrollo basado en plantillas también se encuentran descritos en Recio-García et al. (2007a) y Recio-García et al. (2009).

- Otro problema a resolver es el proceso de recuperación de plantillas. En nuestra aproximación, el usuario recuperará una plantilla en base a las tareas que contenga y los métodos que resuelvan estas tareas. Las medidas de similitud utilizadas también tendrán en cuenta otros factores que se detallarán en la Sección 6.3 junto con todo el mecanismo de recuperación planteado (Recio-García et al., 2008a).

- Finalmente aparece la cuestión de la adaptación de las plantillas. Normalmente sólo será necesario modificar los métodos asignados a algunas de las tareas. Sin embargo este proceso puede ampliarse y habrá ocasiones donde el usuario necesitará modificar las plantillas añadiendo y eliminando tareas, o modificando el flujo de control. También podrá ser necesario implementar nuevos métodos. Estos procesos de adaptación se detallarán en la Sección 6.5 y han sido presentados en Recio-García et al. (2009).

El grado en que se necesiten adaptaciones más o menos radicales dependerá directamente de la cobertura del espacio de posibles diseños CBR que ofrezca la librería de plantillas (que es nuestra base de casos). De esta forma llegamos a la cuestión fundamental de si el CBR es válido para diseñar

sistemas software en general y sistemas CBR en particular. Es decir, sí los problemas (diseños) tienden a recurrir y si los diseños similares tienen soluciones (implementaciones) similares. Esta pregunta era la premisa inicial se planteó antes de desarrollar toda la aproximación presentada en este capítulo. De forma intuitiva, nuestra propia experiencia en el dominio de las aplicaciones CBR sugería que dicha pregunta tenía respuesta afirmativa, que los sistemas CBR tienden a repetir el mismo comportamiento y que un experto podría abstraer dicho comportamiento en un número relativamente bajo de plantillas. Después de la implementación de las herramientas para el diseño basado en plantillas se contrastó de forma experimental esta hipótesis inicial demostrando que nuestra suposición era acertada. Esta evaluación se presenta en el Capítulo 7, donde se compara el mecanismo de diseño basado en plantillas frente a la implementación de sistemas mediante la utilización directa de los métodos del armazón en su capa de caja blanca (diseño basado en métodos). Por lo tanto, en el capítulo actual únicamente se realizará una descripción de la aproximación desarrollada, dejando su evaluación para dicho Capítulo 7.

El problema más importante de los planteados a la hora de afrontar el desarrollo de nuestra aproximación CBR para el diseño de sistemas CBR es la adquisición de las plantillas. Este proceso inicial debe realizarse por expertos que sepan como abstraer el comportamiento de los sistemas CBR. Así que nosotros hemos empezado a poblar la librería con plantillas de sistemas de recomendación que sirvan como prueba de concepto. En el siguiente apartado se incluyen los detalles de la organización y representación de dicha librería de plantillas.

6.2 La librería de plantillas

De forma genérica, las plantillas de jCOLIBRI 2 se componen de tareas que identifican los distintos pasos del sistema CBR y métodos que resuelven dichas tareas mediante una implementación particular. Según la arquitectura de jCOLIBRI 2 (Figura 3.11), estos métodos se encuentran en la parte inferior (de caja blanca) del armazón por lo que será necesario añadirles una representación semántica que permita definir su comportamiento y las tareas que resuelven para poder utilizarlos en el diseño basado en plantillas. Este mecanismo para incorporar la información semántica a los métodos se detallará en la Sección 6.3.3.

La librería de plantillas de nuestro armazón se organiza de forma jerárquica a partir de los siguientes tres niveles:

- En el nivel más bajo encontramos las *plantillas ejecutables* o *sistemas*. En este tipo de plantillas sólo aparecen *tareas simples*, que son aquellas

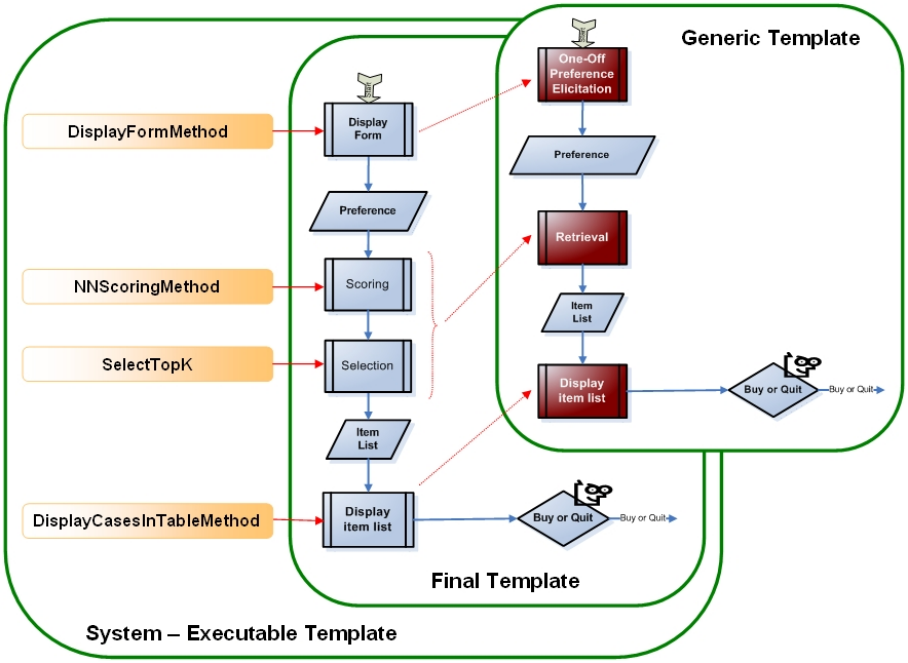


Figura 6.1: Ejemplo de organización de la librería de plantillas

que pueden ser resueltas directamente por un método de la librería. Además, en las *plantillas ejecutables* todas las tareas simples tienen asignado un método que las resuelve. Por lo tanto, este tipo de plantillas representan sistemas completamente funcionales.

- Los *sistemas* del nivel anterior se organizan de acuerdo a la *plantilla final* que implementan. Las plantillas finales estarán compuestas por *tareas simples* pero su representación no incluye los métodos que resuelven cada una de ellas. De esta forma, una plantilla final podrá ser instanciada en varios *sistemas* dependiendo de los métodos asignados para resolver cada tarea simple.

- En el nivel más alto aparecen las *plantillas genéricas* que ofrecen una visión de alto nivel de un conjunto de plantillas finales. Estas plantillas estarán formadas por *tareas genéricas* y tareas simples. Al igual que en el nivel de los sistemas, las tareas genéricas encapsulan secuencias de tareas simples. Dependiendo de la descomposición de cada tarea genérica en secuencias de tareas simples obtendremos varias plantillas finales. Por lo tanto, las plantillas genéricas representan familias de plantillas finales ofreciendo una descripción de alto nivel de dichas familias.

La Figura 6.1 resume mediante un ejemplo la organización de la librería de plantillas. En el nivel más alto (arriba a la derecha) se encuentra una plantilla final que representa un sistema de recomendación. Los detalles de este tipo de sistemas de recomendación se incluyen en el siguiente subapartado, por lo que de momento nos centraremos únicamente en su estructura. La plantilla genérica incluye una serie de tareas genéricas mostradas en rojo. A grandes rasgos la plantilla define un esquema donde se obtienen las preferencias del usuario ("One-Off Preference Elicitation"), se recuperan los artículos más similares ("Retrieval") y se muestran al usuario ("Display item List") para que decida si quiere comprar uno de los artículos propuestos o salir. Esta plantilla genérica representa a alto nivel una serie de plantillas finales donde las tareas genéricas son descompuestas en secuencias de tareas simples (que serán representadas en las figuras mediante el color azul). En la figura se muestra una posible plantilla final donde la tarea genérica "One-Off Preference Elicitation" se ha descompuesto en la tarea simple "Display Form" (mostrar formulario). Por su parte, la tarea genérica "Retrieval" aparece descompuesta en una secuencia formada por las tareas simples "Scoring" (puntuar) y "Selection" (seleccionar). Finalmente la tarea "Display item list" se descompone en una tarea simple con el mismo nombre. Si asignamos métodos que resuelvan cada una de las tareas simples de la plantilla final obtenemos el sistema que se muestra en la parte izquierda de la Figura 6.1. En este caso, la tarea de mostrar formulario se resuelve con el método `DisplayFormMethod`, la tarea de puntuación mediante el método del vecino más próximo (`NNScoringMethod`), la tarea de selección mediante la elección de los k mejores (`SelectTopK`), y la tarea encargada de mostrar los artículos se resuelve mediante un método que los presenta en una tabla (`DisplayCasesInTableMethods`). La librería de métodos incluye diversas implementaciones para resolver las tareas simples, por lo que es posible diseñar distintos sistemas a partir de una plantilla final. A su vez, dependiendo de la secuencia de tareas simples elegida para descomponer una tarea genérica se obtienen diversas plantillas finales. De esta forma, la organización de plantillas propuesta permite representar un amplio abanico de sistemas CBR de una forma jerárquica y bien estructurada. Como caso de estudio, en el siguiente apartado se describe una librería de plantillas formada por plantillas de sistemas de recomendación.

6.2.1 Plantillas de sistemas de recomendación

Actualmente existe una sobrecarga de información que dificulta la búsqueda productos o servicios en grandes catálogos informáticos –ya sea la Web en general o bases de datos locales. Cuando un usuario realiza una búsqueda en este tipo de bases de entornos suele encontrar demasiados productos que no cumplen con sus preferencias y que le hacen perder el interés en dicha búsqueda. Como solución a este problema, los sistemas de recomendación utilizan los objetivos y preferencias del usuario para filtrar la información y seleccio-

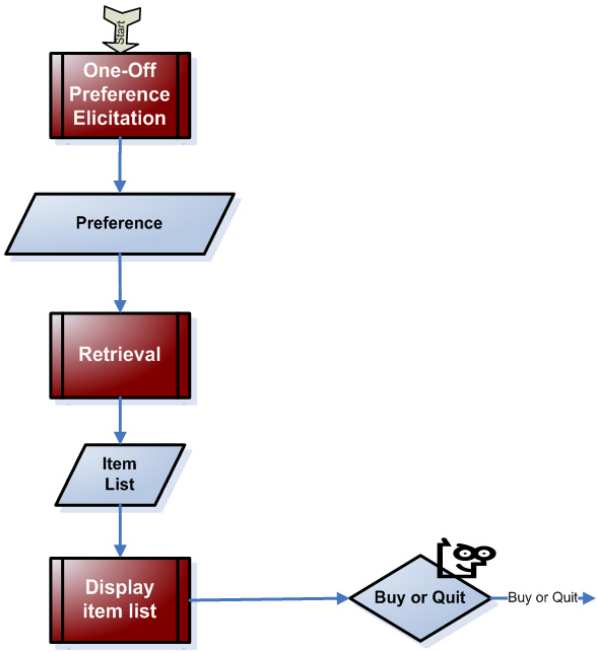


Figura 6.2: Sistemas de recomendación única (Single-Shot)

nar productos, servicios o fuentes de información que pueden ser interesantes para el usuario (normalmente denominados con el término inglés *items* que nosotros traduciremos como *artículos*). Estos sistemas combinan ideas de las áreas de Recuperación de Información, Filtrado de Información, Modelado de Usuarios, Aprendizaje Máquina e Interacción Humano-Máquina.

La mayoría de los sistemas de recomendación se basan en "dialogar" con el usuario para refinar sus preferencias y así poder recomendar el artículo que más se ajuste a ellas. Por lo tanto, los recomendadores pertenecerían al área de los sistemas CBR conversacionales. De hecho, el CBR ha jugado un papel muy importante en el desarrollo de varios tipos de recomendadores tal y como se expone en Bridge et al. (2006) y Smyth (2007).

Los recomendadores son la primera familia de sistemas CBR incorporados a la librería de plantillas de jCOLIBRI. Para obtener estas plantillas se realizó un análisis de sistemas de recomendación que se basaba en la clasificación conceptual descrita en Bridge et al. (2006). Esta clasificación distingue entre recomendadores colaborativos y basados en casos, reactivos y proactivos, de recomendación única (single-shot) o conversacionales, y basados en preguntas o en propuestas. Dentro de esta clasificación, los autores revisan una selección de artículos de la bibliografía sobre sistemas recomendadores basados en casos que abarca el desarrollo de este tipo de aplicaciones durante los últimos diez años.

La construcción de la librería de plantillas se basó en el comportamiento de interacción con el usuario de los sistemas de recomendación. A partir de este criterio se generaron las siguientes plantillas genéricas:

1. Los sistemas de recomendación única (más conocidos por su término inglés Single-Shot) reciben una descripción de las preferencias del usuario, realizan una sugerencia y terminan. La Figura 6.2 muestra la plantilla para este tipo de sistemas, donde las tareas "One-Off Preference Elicitation" (adquisición inicial de preferencias), "Retrieval" (recuperación) y "Display Item List" (mostrar lista de artículos) son tareas genéricas que serán descompuestas en secuencias de tareas simples para dar lugar a un conjunto de plantillas finales y sistemas.

2. Una vez que se han recuperado los artículos, los Sistemas Conversacionales (cuyas plantillas aparecen en las Figuras 6.3 y 6.4) invitan al usuario a refinar sus preferencias actuales, normalmente a través de los artículos recomendados. Por lo tanto, existen diferentes formas de descomponer y resolver la tarea "Iterated Preference Elicitation" (adquisición iterativa de preferencias). Una primera aproximación consiste en permitir al usuario que seleccione un artículo y lo critique para modificar la consulta actual del sistema. Las *críticas* del usuario pueden consistir tanto en definir un valor específico para uno de los atributos del artículo seleccionado (ej.: como este piso pero con 4 habitaciones), como en definir una variación en un atributo (ej.: como este piso pero más barato). De forma alternativa, el refinamiento de las preferencias del usuario puede realizarse preguntando al usuario ciertas preguntas sobre la consulta (por ejemplo, el valor de un atributo para descartar los elementos donde no esté presente).

   La plantilla mostrada en la Figura 6.3 muestra el esquema utilizado cuando se realiza la adquisición iterativa de preferencias mediante selección de artículo más crítica. En este caso siempre se propone algún artículo en cada iteración y por lo tanto se presupone que el conjunto de artículos recuperado (en la tarea "Retrieval") nunca será vacío.

   Sin embargo cuando la tarea "Iterated Preference Elicitation" se resuelve mediante preguntas sobre la consulta, los artículos recuperados pueden mostrarse al usuario en cada iteración (Figura 6.3) o cuando se cumpla un cierto criterio, por ejemplo, que el tamaño del conjunto de artículos recuperado tenga un tamaño máximo. Esta última opción es la que corresponde a la plantilla genérica que aparece mostrada en la Figura 6.4. Hay que destacar que ambas plantillas comparten con los sistemas Single-Shot las tareas "One-off Preference Elicitation" y "Retrieval", junto con los métodos asociados a las mismas.

Dependiendo de la descomposición en tareas simples de las diferentes tareas genéricas de estas tres plantillas se obtienen una gran variedad de

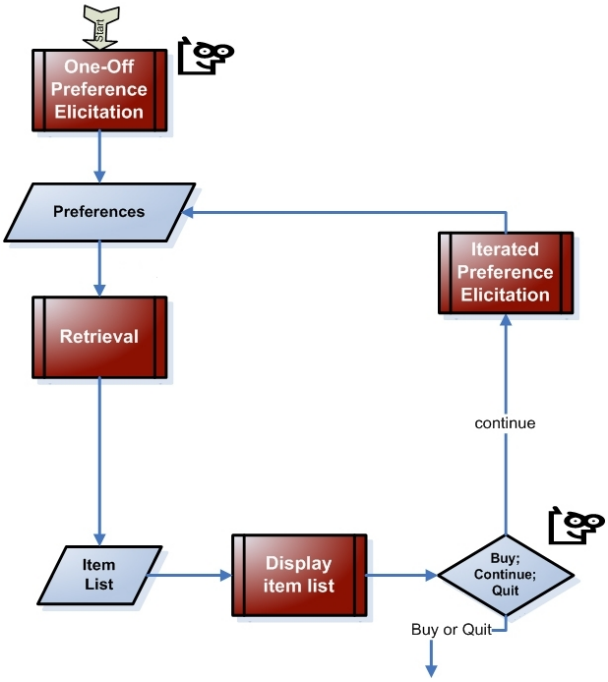


Figura 6.3: Sistemas Conversacionales tipo A

plantillas finales. Estas plantillas finales podrán ser luego instanciadas en sistemas (plantillas ejecutables) dependiendo de los métodos asignados para resolver sus tareas simples. A continuación se presentan las distintas descomposiciones de las tareas genéricas presentes en las plantillas de las Figuras 6.2, 6.3 y 6.4.

6.2.1.1 Adquisición inicial de preferencias

Existen disintas descomposiciones en tareas simples de la tarea encargada de obtener las preferencias iniciales del usuario (One-Off Preference Elicitation). Muchas de estas descomposiciones pueden aplicarse también para resolver la tarea de adquisición iterativa de preferencias (Iterated Preference Elicitation). La Figura 6.5 muestra las descomposiciones de más alto nivel de dichas tareas mientras que la Figura 6.6 detalla la descomposición en tareas simples de las tareas genéricas restantes. En nuestra librería encontramos las siguientes descomposiciones para la tarea *Adquisición inicial de preferencias* (One-Off Preference Elicitation):

- Identificación mediante perfiles ("Profile Identification", Figura 6.5). Aquí, el usuario se identifica en el sistema, lo que permite recuperar su perfil de una base de datos de perfiles. Este perfil podría estar basado

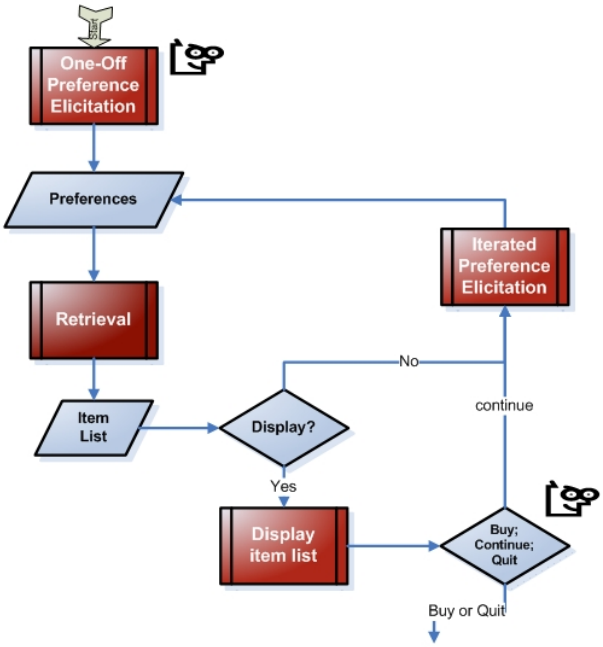


Figura 6.4: Sistemas Conversacionales tipo B

en contenidos (por ejemplo: palabras clave describiendo los intereses anteriores del usuario, descripciones de los artículos consumidos previamente, o descripciones de las sesiones que el usuario ha mantenido con el sistema). Otra opción es que el perfil sea colaborativo y contenga valoraciones de los productos que podrán utilizarse para recomendar artículos a usuarios similares. Esta descomposición en tareas simples también permite la posibilidad de que el usuario sea nuevo en el sistema, por lo que se añadiría un proceso previo de registro antes de la tarea encargada de recuperar el perfil. El detalle de la secuencia de tareas se muestra en la parte superior de la Figura 6.6 (las líneas punteadas en el borde de una tarea simple significa que es opcional).

- Adquisición inicial de la consulta ("Initial Query Elicitation'). Esta opción admite distintas descomposiciones según aparece en la parte central de la Figura 6.5. Las tareas genéricas resultantes se detallan en la Figura 6.6. Entre estas descomposiciones encontramos la secuencia "Form-Filling", es decir, rellenar un formulario. Otra opción denominada "Navigation-by-Asking" consiste elegir una pregunta sobre la consulta y planteársela al usuario. Otra forma de descomponer la tarea "Initial Query Elicitation" la encontramos en las numerosas versiones del sistema Entrée de Burke (2002) donde se utilizan varios métodos

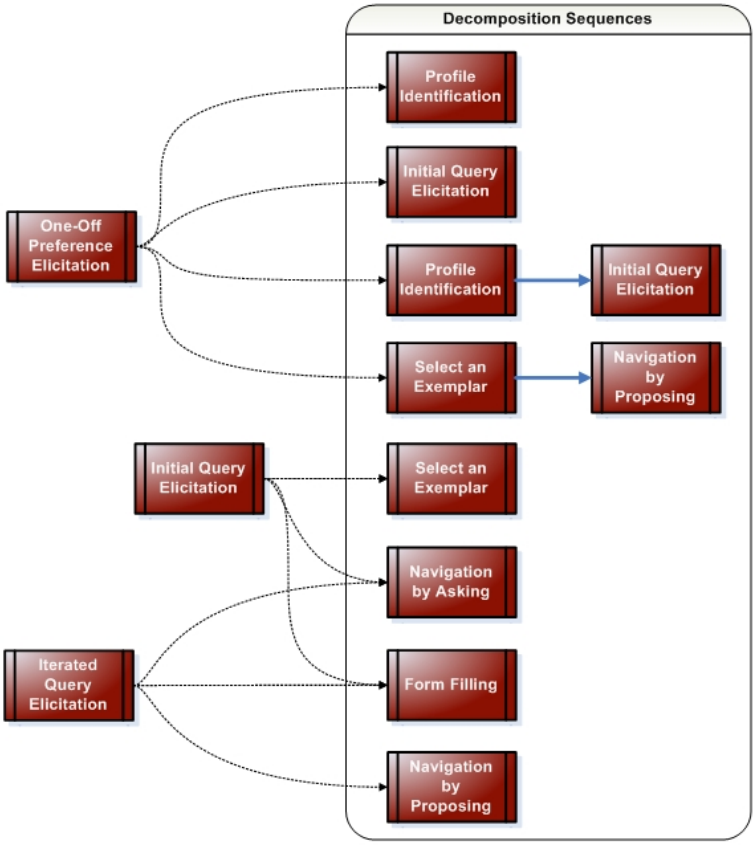


Figura 6.5: Descomposición inicial de las tareas de adquisición preferencias inicial e iterativa

para identificar o seleccionar un artículo. La primera opción consiste en que el usuario introduce información suficiente para identificar un artículo (por ejemplo un restaurante en su ciudad) y esa descripción sirve como base de una consulta (por ejemplo, de un restaurante en otra ciudad). La otra opción se basa en mostrar un conjunto de artículos suficientemente variado y completo como para cubrir el espectro de posibles artículos del sistema. El usuario elegiría entonces el artículo buscado, y si no se encontrara en la lista, seleccionaría el artículo más parecido. Este artículo serviría como consulta para la siguiente iteración con el sistema. Ambas descomposiciones –identificar un artículo y seleccionar un artículo– pueden descomponerse en una misma secuencia que se muestra en la Figura 6.6 como "Select an Exemplar". Dependiendo del método asignado para resolver la tarea de seleccionar artículos ("Items Selecction") obtendríamos un comportamiento u otro, ya que si seleccionamos artículos que el usuario ya conoce estaríamos

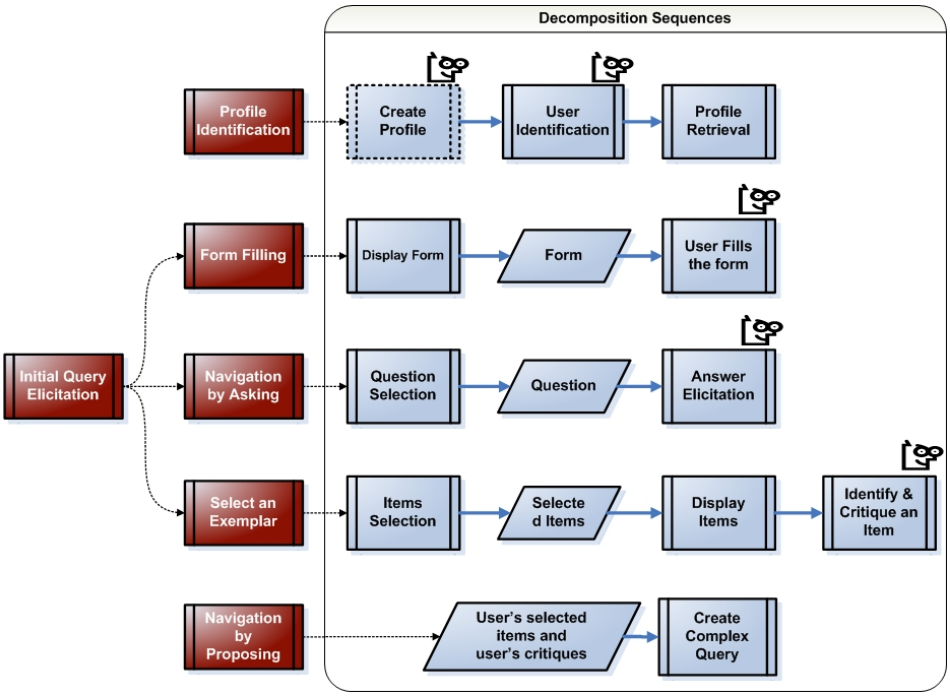


Figura 6.6: Descomposición detallada de las tareas de adquisición preferencias inicial e iterativa

en la opción de identificar un artículo y si elegimos un conjunto de artículos diversos implementaríamos la opción de seleccionar un ejemplar.

- La tercera posibilidad consistiría en unir las dos anteriores (identificación mediante perfil y adquisición inicial de la consulta) donde se combinarían las dos tareas descritas previamente. Esta descomposición aparece en la Figura 6.5.

6.2.1.2 Recuperación

La tarea de *Recuperación* es común a todas nuestras plantillas, y por lo tanto es la más importante. Esta tarea genérica permite varias descomposiones alternativas. En general, la descomposión a utilizar dependerá de la elección tomada al descomponer otras tareas de la plantilla. Por ejemplo, si la adquisición inicial de preferencias devuelve una serie de perfiles sobre valoraciones de productos, entonces el método elegido para resolver la tarea de recuperación deberá implementar algún tipo de recuperación colaborativa.

En la mayoría de las descomposiciones, la recuperación es un proceso dividido en dos pasos donde encontramos una primera etapa de puntuación

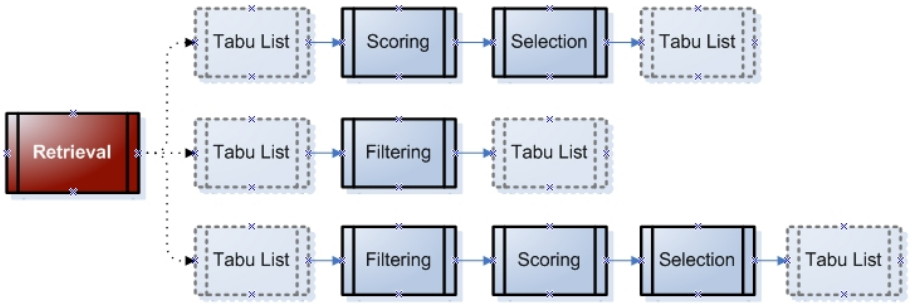


Figura 6.7: Descomposición de la tarea genérica de recuperación.

de los casos ("Scoring") seguida de otra etapa de selección. De cualquier modo, existen otras descomposiciones de la tarea genérica de recuperación que no siguen este esquema de dos pasos. Un ejemplo es la recuperación por filtro ("Filter-Based Retrieval"), donde las preferencias del usuario son tratadas como restricciones que los artículos recuperados deben cumplir. Además, existen otros sistemas de recomendación donde la recuperación se descompone en más de dos etapas. Por ejemplo, en algunos tipos de recomendadores que siguen la aproximación "Navigation-by-Proposing" (detallada en el próximo apartado), se obtiene inicialmente un conjunto de artículos, el usuario elige el más parecido al artículo buscado y lo critica. Dichas críticas se utilizan como restricciones en la recuperación por filtro y, a continuación, los artículos filtrados son puntuados según su similitud con un artículo seleccionado por el usuario. Finalmente se muestra al usuario el subconjunto de casos que cumplen las restricciones y tienen una puntuación más alta.

La Figura 6.7 muestra un diagrama con algunas descomposiciones posibles de la tarea genérica de Recuperación. En esta figura aparecen de forma opcional las tareas simples para manejar listas de artículos tabú. Estas listas almacenarían los artículos que ya han sido presentados al usuario para evitar mostrarlos varias veces. Antes de realizar la recuperación se borrarían los artículos que hubieran sido mostrados anteriormente para después añadir los artículos recuperados en la iteración actual a la lista tabú.

6.2.1.3 Adquisición iterativa de preferencias

En la adquisición iterativa de preferencias ("Iterated Preference Elicitation") el usuario, a quien puede haberse mostrado o no algunos productos previamente (Figura 6.4), tiene la posibilidad de indicar más información sobre sus preferencias. Esta acción puede ser voluntaria o requerida por el sistema, y ayuda a mejorar los resultados de la siguiente iteración en la conversación recomendador-usuario. Varias descomposiciones de esta tarea son comunes con la adquisición inicial de preferencias (ver Figuras 6.5 y 6.6). Las dis-

tintas descomposiciones de la tarea "Iterated Preference Elicitation" son las siguientes:

- Rellenar un formulario ("Form Filling") donde el usuario introduce sus preferencias de la misma forma en que se representan los artículos en la base de casos/artículos. Es decir, se especifican valores para los distintos atributos del artículos.

  Como ya hemos visto, la tarea "Form Filling" también se utiliza en la adquisición inicial de preferencias. Cuando se incluye también para resolver la tarea de adquisición iterativa, normalmente el usuario edita valores que hubiera introducido anteriormente en un formulario inicial.

- La navegación mediante preguntas ("Navigation-by-Asking") es la otra descomposición que puede utilizarse para resolver ambas tareas genéricas (One-Off y Iterated Preference Elicitation). Aquí el sistema refina la consulta mediante las respuestas del usuario a preguntas sobre sus preferencias. El sistema utiliza heurísticas para elegir la siguiente mejor pregunta a plantear. Estos algoritmos se proponen originalmente en el trabajo de Bergmann (2002).

- La navegación mediante propuestas ("Navigation-by-Proposing") requiere que se hayan mostrado al usuario un conjunto de artículos candidatos. El usuario seleccionará aquel artículo que se parezca más a sus requisitos y además ofrecerá una serie de críticas (p.e. "como esto pero más barato"). Utilizando esta información, el sistema construirá una nueva consulta que permita recuperar artículos similares al seleccionado pero que además cumplan los requisitos impuestos por las críticas. Esta descomposición se muestra en la parte inferior de las Figuras 6.5 y 6.6.

  Podemos encontrar trabajos iniciales sobre esta aproximación en Burke (2002). Aunque desde entonces han aparecido muchas aportaciones a esta idea. Se puede encontrar una descripción detallada en Bridge et al. (2006) y Smyth (2007).

  Aunque nosotros sólo describimos la tarea "Navigation-by-Proposing" como una posible descomposición de la adquisición iterativa de preferencias, esto no tiene por que ser estrictamente así. Si, por ejemplo, combinamos la navegación mediante propuestas con la selección de un artículo ("Select an Exemplar") obtendríamos una descomposición apropiada para resolver la tarea de adquisición inicial de preferencias. En esta secuencia se presentarían inicialmente una serie de artículos diversos y el usuario elegiría uno de ellos (junto con críticas sobre él) para hacer de consulta inicial del sistema. Esta aproximación se muestra en las descomposiciones de la tarea "One-Off Preference Elicitation" que aparecen en la Figura 6.5.

Cabe destacar que las plantillas de Sistemas Conversacionales no excluyen la posibilidad de que el sistema utilice métodos diferentes para resolver la adquisición iterativa de preferencias en diferentes iteraciones. ExpertClerk es un ejemplo de este tipo de recomendadores. Este sistema utiliza navegación mediante preguntas para la adquisición inicial de preferencias. A continuación elige entre navegación mediante preguntas o navegación mediante propuestas para la tarea de adquisición iterativa de preferencias. Esta elección se toma dependiendo del tamaño de artículos recuperado (cuando desciende a un tamaño predefino se cambia de preguntas a propuestas). Los detalles sobre este sistema aparecen en Shimazu (2002). Nosotros hemos confirmado que es posible construir dicho sistema a partir de la plantilla mostrada en la Figura 6.4 y los métodos que se describirán en el siguiente apartado. Sin embargo, por ciertas razones redundantes en esta discusión, encontramos que se necesitaba una descomposición especial de la tarea para mostrar los artículos ("Display Item List") en la plantilla de la Figura 6.4. Es por ello que esta tarea es genérica en vez de simple. Excepto en la construcción de este sistema, la tarea genérica de mostrar la lista de casos siempre se descompondrá en una única tarea simple.

La posibilidad de implementar un sistema tan complejo como ExpertClerk mediante nuestra librería de plantillas demuestra informalmente las posibilidades de nuestra aproximación: sistemas complejos pueden ser construidos adaptando plantillas simples existentes.

6.2.2 Métodos para instanciar las plantillas de recomendadores

Las plantillas descritas en la sección anterior han sido incluidas en jCOLIBRI y pueden ser utilizadas por los diseñadores de sistemas CBR. En esta sección detallaremos los diferentes métodos del armazón que pueden asignarse para resolver cada una de las tareas simples de las diferentes plantillas.

En jCOLIBRI se han incluido varias *plantillas finales* donde las tareas complejas de las plantillas genéricas se descomponen en tareas simples. De esta forma se han obtenido 10 plantillas que representan una gran variedad de recomendadores prototípicos. Cada una de estas tareas se ha resuelto mediante uno de los métodos aquí descritos para obtener *sistemas* que sirvan de ejemplo a los desarrolladores a la hora de diseñar nuevos sistemas de recomendación. De esta forma, los sistemas incluidos en el armazón siguen el comportamiento definido por las plantillas y sirven para ejemplificar el código completo de un recomendador real[1]. Actualmente, el armazón incluye 10 plantillas finales y 14 sistemas de ejemplo implementados a partir de dichas plantillas.

[1]La descripción completa de los ejemplos y plantillas incluidos en jCOLIBRI puede accederse a través de: http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2/doc/api/jcolibri/test/recommenders/package-summary.html

Los métodos de jCOLIBRI para la implementación de sistemas de recomendación se organizan según la tarea simple que resuelven. Por ejemplo, existen varios métodos para implementar la tarea de selección de casos ("Cases Selection") que pueden ser intercambiados entre ellos al implementar una plantilla. Muchos de estos métodos pueden utilizarse en sistemas CBR genéricos y ya pertenecían a la biblioteca de métodos antes de crear la extensión para recomendadores del armazón. Por lo tanto, se reaprovecha en gran medida funcionalidad que ya estaba incluida en jCOLIBRI. A continuación describiremos brevemente estos métodos aunque hay que resaltar que cualquier desarrollador podría añadir sus propios métodos al armazón y utilizarlos para instanciar una plantilla. También hay que tener en cuenta que la siguiente recopilación no incluye los métodos relacionados con la interfaz gráfica como los encargados de mostrar listas de artículos o preguntas.

6.2.2.1 Métodos de recuperación

La tarea de recuperación obtiene y puntúa los artículos que serán presentados al usuario. Esta tarea puede ser resuelta por los siguientes métodos:

- Recuperación basada en similitud mediante el algoritmo del vecino más próximo (*Nearest Neighbor*). Además de las medidas de similitud incluidas en el armazón para sistemas CBR genéricos, se incorporan nuevas medidas específicas de los sistemas de recomendación adaptadas de McSherry (2003) y Vollrath et al. (1998).

- Recuperación por filtro. Recupera los artículos cuyos atributos cumplen una serie de restricciones. Los desarrolladores pueden definir estas restricciones mediante *predicados* (operadores booleanos como igual, menor que, mayor o igual que,...) que comparan los valores de un atributo en la consulta y en el artículo. Si todas las restricciones se cumplen entonces el artículo es incluido en el conjunto recuperado. Como ejemplo, podemos pensar en un sistema de recomendación de pisos donde el usuario indique en su consulta el precio máximo de la vivienda. Para obtener únicamente los pisos que cumplan esa restricción utilizaremos una recuperación por filtro con la condición "menor que" sobre el atributo "precio".

- Recuperación a partir del caso medio. Esta método es una adaptación del algoritmo implementado en el sistema ExpertClerk de Shimazu (2002). Este método devuelve casos diversos sin necesidad de consulta. Para ello, se crea un artículo artificial que representa la mediana de todos los artículos presentes en la base de casos. El "caso medio" se obtiene haciendo la media de cada uno de los atributos de todos los casos. Si los atributos no son numéricos se utilizará el valor que se repita más veces. A partir de este artículo imaginario se recupera un

número determinado de artículos teniendo en cuenta cuantos atributos se diferencian superando un cierto límite con respecto al caso medio. Los artículos recuperados serán aquellos que tengan más atributos superando los límites establecidos para cada atributo. Este método es muy útil para proponer artículos diversos en la etapa de adquisición inicial de preferencias ("One-Off Preference Elicitation").

- Recuperación colaborativa. Este método sigue el esquema propuesto en Kelleher y Bridge (2004); Herlocker et al. (1999) para recuperar artículos en función de las valoraciones o votos indicados por los distintos usuarios del sistema:

  1. Ordenar todos los usuarios del sistema respecto a su similitud con el usuario activo. Para ello se utiliza el algoritmo de Correlación de Pearson (Moore, 1999).
  2. Seleccionar el subconjunto de usuarios más parecidos cuyas valoraciones se utilizarán para realizar la recomendación.
  3. Normalizar las valoraciones de los usuarios y calcular la recomendación mediante una combinación ponderada de dichos votos.

  Este método utiliza métodos auxiliares para manejar perfiles de los usuarios donde se guarden las valoraciones introducidas en el sistema.

6.2.2.2 Métodos de selección

La tarea de selección de casos se encarga de seleccionar uno o más artículos que serán mostrados al usuario final del conjunto devuelto por el método de recuperación. Para ello se ofrecen los siguientes métodos:

- Seleccionar todos o los k mejores casos del conjunto recuperado (según la medida de similitud utilizada en la recuperación).

- Selección dirigida por el compromiso ("Compromise driven selection" de McSherry (2003)). Este algoritmo selecciona los artículos que minimizan sus compromisos con la consulta del usuario. Los compromisos son aquellos atributos que no satisfacen los requisitos del usuario expresados en la consulta. De este modo, los casos se seleccionan minimizando el número de compromisos y maximizando su similitud con la consulta.

- Selección Voraz ("Greedy Selection" de Smyth y McClave (2001)). Este método considera tanto la similitud como la diversidad de los artículos recuperados. Para ello se construye incrementalmente un conjunto de artículos añadiendo aquellos cuya calidad sea la mayor respecto a los demás artículos ya seleccionados. La calidad de un artículo es proporcional a su similitud a la consulta y a su diversidad relativa con los artículos seleccionados. Aquí, la diversidad es la inversa de la similitud.

Debido al alto coste en tiempo de este método, existe una modificación denominada "Bounded Greedy Selection" (Selección Voraz Limitada) donde se restringe el proceso de selección a un subconjunto de artículos más reducido que se elige aleatoriamente.

6.2.2.3 Métodos de Navegación mediante Preguntas

Existen diversas posibilidades para elegir cual es la siguiente mejor pregunta que debe plantearse al usuario cuando se utiliza la estrategia de navegación mediante preguntas ("Navigation by Asking"). Estas heurísticas seleccionan el atributo más relevante que discrimine más artículos de la base de casos. Los métodos implementados en jCOLIBRI son:

- Ganancia de información. Devuelve el atributo con mayor ganacia de información en un conjunto de artículos (Schulz, 1999; Bergmann, 2002).

- Influencia en la similitud ("Similarity Influence Measure" de Bergmann (2002)). Selecciona el atributo que tiene la mayor influencia en el cálculo del vecino más próximo (k-NN). Esta influencia en la similitud puede medirse a partir de la varianza de todas las similitudes en un conjunto de artículos.

6.2.2.4 Métodos de Navegación mediante Propuestas

La modificación de las preferencias del usuario de acuerdo a sus críticas a artículos propuestos por el sistema puede realizarse mediante diferentes estrategias propuestas por McGinty y Smyth (2002):

- Más como este (*MLT* de "More Like This"). Remplaza la consulta actual del sistema con la descripción del artículo seleccionado por el usuario. Aunque es una estrategia muy común tiene el problema de que quizás el usuario no deseaba alguno de los atributos presentes en el artículo seleccionado.

- Más como este parcial (*pMLT* de "Partial More Like This"). Remplaza parcialmente la consulta actual a partir de los atributos del artículo seleccionado. Por ejemplo, en la Tabla 6.1 aparece un ejemplo sobre pisos donde el caso seleccionado sí está amueblado (caso 3). De acuerdo a la estrategia MLT, dicho valor debería ser copiado a la consulta. Sin embargo, el caso 2 también está amueblado y ha sido rechazado. Esto puede significar que el usuario no estaba interesado en el valor de ese atributo cuando eligió el caso 3. Esta idea es la que se captura en la estrategia pMLT ya que sólo transfiere el valor de un atributo a la consulta si ninguno de los artículos rechazados por el usuario tiene el mismo valor que dicho atributo. Por lo tanto, en el ejemplo

anterior transferiríamos a la consulta los valores de los atributos área, habitaciones, precio y baños, ya que no están repetidos en los demás pisos.

- Más como esto ponderado (*wMLT* de "Weighted More Like This"). Volviendo a la Tabla 6.1, ¿con qué seguridad debemos transferir el valor "El Pilar" a la consulta? ¿Debemos descartar completamente que el piso sí esté amueblado simplemente porque se ha rechazado otro piso que también lo está? La estrategia wMLT intenta tener en cuenta estas preguntas y transfiere todos los atributos desde el artículo seleccionado a la consulta pero ponderandolos al dar preferencia a los atributos más diversos entre los casos propuestos. La idea es que podremos estar más seguros de la decisión del usuario respecto a un atributo cuantas más alternativas se le hayan presentado. De esta forma se indica que el valor "El Pilar" es muy seguro porque el usuario no ha rechazado ningún otro piso que tenga dicho valor. Por contra, el valor "Sí" del atributo "Amueblado" tendrá menos valor porque se encuentra rechazado en otro caso. La implementación de este método realiza la ponderación de los atributos modificando los pesos del algoritmo del vecino más próximo (que deberá utilizarse en la recuperación).

- Menos como esto (*LLT* de "Less Like This"). Es una estrategia bastante simple: si todos los artículos rechazados por el usuario tienen el mismo valor en un atributo y ese valor es distinto al del atributo del artículo seleccionado, entonces se añade el valor rechazado como una condición negativa para el atributo en cuestión. Dicha condición negativa se representa en forma de un predicado para el método de recuperación por filtro (que deberá utilizarse para resolver la tarea de recuperación). En el ejemplo de la Tabla 6.1 podemos observar que todos los casos rechazados tienen un solo baño mientras que el seleccionado tiene dos. Por lo tanto la estrategia LLT añadiría la restricción de que el número de baños sea siempre distinto de 1.

- Más y Menos como esto (*M+LLT* de "More + Less Like This"). Combina las dos estrategias *MLT* y *LLT* copiando los valores del caso seleccionado en la consulta y añadiendo predicados negativos para el método de recuperación por filtrado. Este método debería utilizarse cuando en la recuperación se combinen los métodos de similitud por el vecino más próximo y filtrado.

6.2.2.5 Métodos condicionales

Estos métodos se encargan de modificar las tareas condicionales como "¿Display?", "¿Continue?" o "¿Buy or Quit?". Los más importantes son los métodos

Atributos	Caso1	Caso2	Caso3	Caso 4
Área	Aluche	Chueca	El Pilar	Vallecas
Habitaciones	2	Estudio	4	3
Precio	200.000	240.000	325.000	190.000
Amueblado	No	Sí	Sí	No
Baños	1	1	2	1

Tabla 6.1: Ejemplo de aplicación de los métodos de navegación mediante propuestas.

que resuelven la tarea condicional que comprueba si el conjunto recuperado debe mostrarse al usuario ("¿Display?"). Esta tarea es muy importante cuando se instancia una plantilla conversacional tipo B (Figura 6.4), y las opciones incluidas en jCOLIBRI son:

- Mostrar el conjunto de artículos sólo si su cardinalidad es superior y/o inferior a unos límites predeterminados. Este método evita presentar al usuario conjuntos de artículos vacíos o demasiado grandes.

- Mostrar el conjunto de artículos sólo si se supera un límite de similitud mínima. De esta forma se evita presentar conjuntos de artículos cuya "calidad" sea demasiado baja ya que se comprueba que todos los artículos superen un límite de similitud con la consulta. Por lo tanto este método es adecuado únicamente cuando se utilice el el método del vecino más próximo en la recuperación.

6.3 Representación semántica de plantillas y métodos

En la sección anterior se han descrito las distintas plantillas y métodos incluidos en jCOLIBRI 2 desde un punto de vista funcional. Sin embargo, no se ha explicado el mecanismo de representación de dichos elementos del armazón. Esta representación es el pilar fundamental de todo el sistema de composición de aplicaciones CBR mediante plantillas ya que debe permitir el razonamiento sobre los distintos componentes involucrados. Por lo tanto, nuestra propuesta consiste en utilizar una *representación semántica* de las plantillas junto con las tareas en que se descompone y los métodos que resuelven dichas tareas. Como se demostrará a lo largo de este capítulo, la representación semántica de los distintos elementos de la plataforma permitirá y facilitará el mecanismo de diseño de sistemas CBR implementando en jCOLIBRI 2.

Para realizar esta representación se ha utilizado un elemento de la plataforma COLIBRI que había quedado delegado a un segundo plano en la

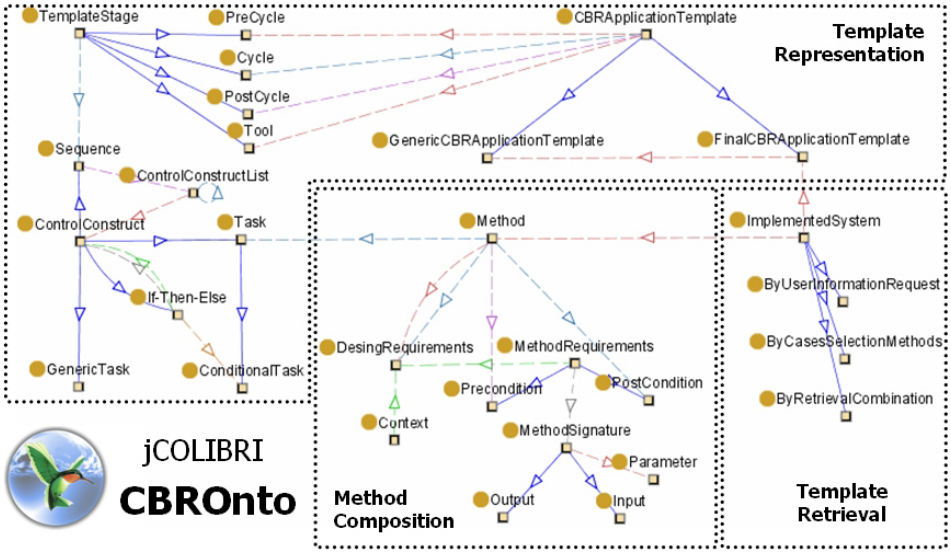


Figura 6.8: Panorámica general de CBROnto en jCOLIBRI 2

versión inicial de jCOLIBRI pero que toma de nuevo una gran importancia en jCOLIBRI 2: la ontología *CBROnto*. Como puede comprobarse en la Figura 3.11 (pág. 94) donde se muestra la nueva arquitectura del armazón, CBROnto es el elemento central de la capa orientada a diseñadores. Según se expuso en la sección 3.1.1 esta ontología sirvió inicialmente para representar el funcionamiento de los distintos elementos de COLIBRI y permitir su composición. Sin embargo, con la cambio al lenguaje Java de jCOLIBRI 1 su importancia disminuye radicalmente al servir únicamente para definir la jerarquía de clases.

En COLIBRI se seguía la metodología KADS donde se definían una serie de tareas que serían resueltas por los métodos de resolución de problemas (PSMs), y toda esta arquitectura se representaba y organizaba a través de CBROnto. Como en jCOLIBRI 2 se sigue manteniendo gran parte de esta arquitectura, parece que reutilizar CBROnto como elemento central de la plataforma es una idea lógica y natural.

Por lo tanto, en el desarrollo de los mecanismos de composición de sistemas CBR presentado en este capítulo, se utilizará CBROnto para representar las plantillas y además guiar los procesos de recuperación y adaptación que se detallarán en los apartados 6.4 y 6.5.

Aunque las etapas anteriores sean bastante diferentes y –en cierto modo– independientes, CBROnto permite centralizar y organizar todo el conocimiento necesario de una forma coherente. Esta organización se muestra en la Figura 6.8 donde aparecen los conceptos más importantes de la ontología.

Las líneas punteadas representan propiedades y las líneas sólidas representan relaciones entre subclases. La figura muestra las tres partes que componen CBROnto, cada una dedicada a un objetivo distinto:

Representación de las plantillas. Esta parte de la ontología contiene los conceptos necesarios para representar las plantillas y se muestra en la parte superior de la Figura 6.8 (bloque "Templates Representation"). Esta parte de CBROnto es una adaptación de varios estándares de Servicios Web Semánticos según se detallará en la Sección 6.3.1. De este modo, aparecen conceptos que definen los pasos de una aplicación CBR (preciclo, ciclo y postciclo), así como su flujo de control (secuencias, condicionales y bucles). El concepto encargado de representar las tareas ("Tasks") juega un papel central en la representación de las plantillas ya que son los componentes básicos de las mismas.

Recuperación de las plantillas. El proceso de recuperación de las plantillas también se aprovecha del conocimiento representado en CBROnto. Para ello se han definido varios conceptos que clasifican las plantillas de acuerdo a ciertas características de los sistemas CBR (dependiendo de la información pedida al usuario, los métodos de selección utilizados, el tipo de navegación, ...). Estos conceptos se muestran en la parte inferior-derecha de la Figura 6.8 (bloque "Template Retrieval"). Gracias a las capacidades de los razonadores de Lógicas Descriptivas podemos clasificar las plantillas bajo estos conceptos y calcular su similitud según se detallará en la Sección 6.4.

Adaptación de las plantillas. La adaptación de la plantilla recuperada consiste en asignar métodos que resuelvan cada una de sus tareas. Cada método requiere de ciertos datos de entrada y salida, y únicamente puede ser ejecutado bajo unas determinadas condiciones. Estos datos y condiciones se representan en la ontología como las entradas/salidas y pre-/postcondiciones del método. Esta representación permite comprobar si un método puede ser utilizado para resolver una tarea dependiendo de los otros métodos asignados a las otras tareas de la plantilla. Para realizar estas comprobaciones se recurrirá a las capacidades de clasificación de las Lógicas Descriptivas según se describirá en la Sección 6.5.

Los conceptos más importantes involucrados en la representación de los métodos se muestran en la parte inferior-central de la Figura 6.8 (bloque "Method Composition") y se detallarán en la Sección 6.3.2

Una vez descrito el marco general de CBROnto, a continuación nos centraremos en las aproximaciones desarrolladas para representar las plantillas y métodos.

6.3.1 Representación de plantillas

La representación de las plantillas utilizadas en jCOLIBRI se basa en los estándares de Servicios Web Semánticos presentados en la introducción de esta memoria (Sección 2.3.2): OWL-S y WSMO que son las dos iniciativas más importantes para la representación y composición de Servicios Web Semánticos. Ambos formalismos tienen en común la utilización de ontologías para representar la información, lo que encaja perfectamente con nuestra aproximación basada en CBROnto. Aunque nuestro lenguaje de representación toma ideas de ambas corrientes, se centra particularmente en el diseño de OWL-S. Esta elección también aprovecha que CBROnto estuviera desarrollada en el lenguaje OWL.

Para ocultar la complejidad del lenguaje OWL al diseñador de las plantillas, jCOLIBRI 2 incluye un editor que permite su creación y edición de forma gráfica. Este editor se muestra en la Figura 6.9. La parte superior muestra una plantilla genérica, mientras que la inferior muestra una plantilla final que extiende la anterior. En la implementación del editor se utiliza la librería OntoBridge (Sección 5.2) que facilita la gestión de la ontología generada por la herramienta y que representa la plantilla creada por el usuario (dicha ontología contendrá una serie de instancias de los conceptos de CBROnto). En principio el editor de plantillas sólo debería utilizarse por el experto en el dominio. Los diseñadores finales únicamente utilizarán las herramientas de recuperación y adaptación que presentamos en las secciones 6.4 y 6.5.

La representación de las plantillas sigue el esquema de OWL-S donde se incluye un perfil del servicio que almacena detalles descriptivos del mismo como el nombre, autor, etc. En el editor de plantillas se permite definir el nombre de la plantilla, autor, naturaleza de la plantilla (genérica o final), tipo o familia a la que pertenece (p.e. single-shot o conversacional), descripción, y etapas auxiliares aparte del preciclo, ciclo y postciclo denominadas *herramientas* adicionales ("Additional Tools") del sistema CBR. Este último elemento se ha incluido para permitir la definición de herramientas de mantenimiento, evaluación o de cualquier otro tipo que pudieran ser necesarias para la aplicación CBR y que se ejecutaran en un orden distinto –quizás en paralelo– que la secuencia preciclo, ciclo y postciclo típica. Este editor de las propiedades de la plantilla se muestra en la parte inferior derecha de la Figura 6.9 y aparece automáticamente al crear una nueva plantilla.

Como puede observarse en la Figura 6.9, cada plantilla se representa mediante las tres etapas de una aplicación CBR en jCOLIBRI: preciclo, ciclo y postciclo (más las herramientas adicionales si las hubiera). Cada una de estas etapas está compuesta por una serie de nodos conectados entre sí. Los nodos

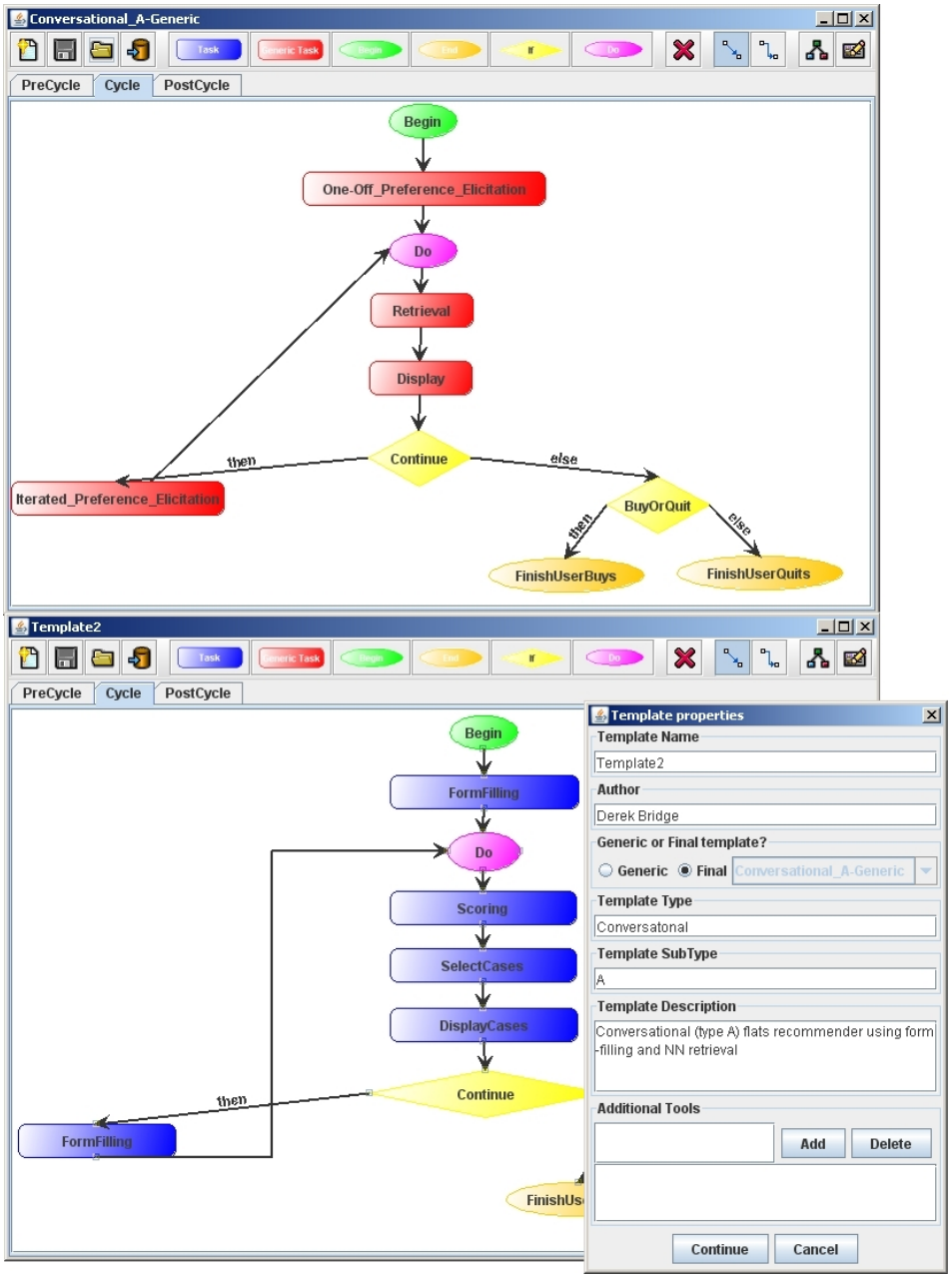


Figura 6.9: Editor de plantillas de jCOLIBRI 2

representan las distintas tareas que componen la plantilla, habiendo diferentes tipos: genéricas (color rojo), simples (color azul) y condicionales (color amarillo). Los dos primeros tipos tareas ya fueron descritas al presentar las plantillas para sistemas de recomendación, sin embargo no se mencionaron las tareas condicionales. Este tipo de tarea es análoga a las tareas simples aunque se le asigna un tipo especial de método que devuelve un valor booleano (método condicional). De esta forma es posible diseñar plantillas con condiciones *if-else*. Además, se han definido una serie de nodos especiales que sirven para indicar el comienzo de una composición (con la etiqueta "Begin" y color verde) y su final (color naranja). También hay un último tipo de nodo especial que permite indicar puntos de retorno para poder implementar bucles (mostrados como "Do" y en color magenta).

En la Figura 6.9 se muestran dos capturas de la herramienta. En la superior se está diseñando una plantilla genérica ya que, como puede observarse, está compuesta de tareas genéricas (en color rojo). En este caso, la plantilla representa a la familia de recomendadores conversacionales de tipo A (cuyo esquema aparecía en la Figura 6.3). La parte inferior muestra una plantilla final que extiende la anterior y donde se han sustituido las tareas genéricas por secuencias de tareas simples (en color azul).

La representación de las plantillas se realiza a través de los conceptos definidos en la ontología CBROnto. Tal y como se muestra en la zona superior de la Figura 6.8, hay una parte de CBROnto encargada de definir la representación de todos los elementos que aparecen en una plantilla junto con sus relaciones. Cuando se diseñe una plantilla, el editor la representará mediante instancias de dichos conceptos de CBROnto. Partiendo del concepto *CBRApplicactionTemplate* el editor generará una instancia que represente la plantilla y que se enlazará con otras instancias de los otros conceptos (PreCycle, Cycle, PostCycle, ...) a través de las propiedades mostradas en dicha figura. Como puede observarse, existen dos subclases del concepto *CBRApplicationTemplate*: *GenericCBRApplicationTemplate* (plantilla genérica) y *FinalCBRApplicationTemplate* (plantilla final). Dependiendo de la elección realizada en la ventana de propiedades, se creará una instancia perteneciente a uno de estos dos conceptos. Cada instancia de *CBRApplicationTemplate* se enlaza mediante una propiedad de la ontología con instancias de los conceptos *PreCycle*, *Cycle*, *PostCycle* y (opcionalmente) *Tool*. Estos conceptos representan las distintas etapas de la aplicación (por ello especializan el concepto *TemplateStage*) y se representan en el editor por cada uno de los tres paneles tabulados. Cada uno de ellos contendrá su propio grafo de ejecución.

Esta representación en CBROnto de las plantillas como secuencias de tareas es una adaptación directa el diseño de servicios compuestos en OWL-S. En este diseño encontramos el concepto *ControlConstruct* que se especializa en secuencias, condiciones y tareas simples, genéricas o condicionales. Cada secuencia estará compuesta por una una lista de construcciones de control

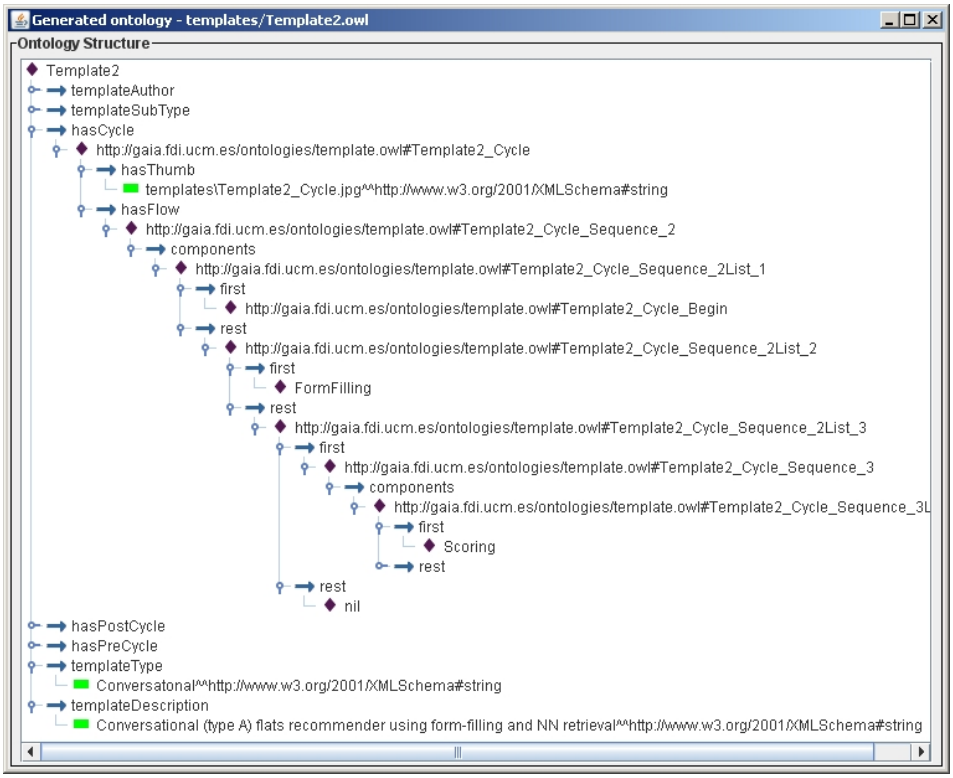


Figura 6.10: Código OWL generado por el editor de plantillas

(*ControlConstructList*) que estará formada a su vez por las subclases del concepto *ControlConstruct*.

El código completo de esta parte de CBROnto se aparece listado en el Apéndice B.1.

Al iniciar la herramienta se carga en memoria el contenido de CBROnto. De esta forma, al crear un nodo de tipo tarea simple o tarea genérica se obtienen las distintas instancias de los conceptos *Task* y *GenericTask* definidas en la ontología y se presentan al diseñador de la plantilla para que las asigne a dicho nodo. La creación de dichas instancias será responsabilidad del experto en el dominio.

Una vez finalizada la edición de la plantilla, la herramienta permite generar el código OWL que la representa siguiendo el formalismo detallado anteriormente. En la Figura 6.10 se puede observar una la representación gráfica del código OWL generado por la herramienta para la plantilla que aparece en la parte inferior de la Figura 6.9. El código completo OWL de la representación de dicha plantilla se incluye en el Apéndice B.2.

6.3.2 Representación de los métodos

La representación semántica del comportamiento de los métodos de jCOLIBRI también se basa en los estándares desarrollados en el ámbito de los Servicios Web Semánticos, tomando ideas tanto de OWL-S como de WSMO. Esta mezcla de representaciones nos permite mantenernos en un punto intermedio entre ambas y nos posibilita una futura compatibilidad con otros componentes software que estuvieren definidos en ambos estándares.

La representación del funcionamiento de los componentes software es actualmente un problema complejo que plantea numerosas dificultades a la hora de abordarlo de forma genérica. Sin embargo, en jCOLIBRI podemos simplificar algunos requisitos de su representación al estar manejando métodos en un entorno bien definido y, en principio, no tan abierto como los servicios web.

A la hora de estudiar la representación semántica de los métodos tenemos que hacer una clara distinción entre dos ámbitos distintos. Por un lado tenemos la representación independiente de cada método, donde sólo se define información sobre sus precondiciones y postcondiciones de ejecución y no se tiene en cuenta información sobre las posibles composiciones de las que forme parte. Por otro lado está la representación de las composiciones de los métodos donde se necesita incluir información sobre el flujo de datos entre las entradas y salidas de los métodos y otras pre-/postcondiciones de la composición generada vista como una única entidad.

En los siguientes apartados se presenta el tipo de representación elegida para los métodos (Sección 6.3.2.1) y las composiciones de métodos (Sección 6.3.2.2). Finalmente, la Sección 6.3.2.2 detalla cómo se ha definido dicha representación a través de CBROnto.

6.3.2.1 Representación de los métodos como entidades independientes

La representación de los métodos en jCOLIBRI 2 sigue un modelo basado principalmente en OWL-S y más específicamente en su *perfil* y *modelo de proceso*. Como se expuso en la Sección 2.3.2.1, el perfil almacena información de alto nivel sobre el servicio como la descripción, autor, etc. y el modelo se encarga de definir su comportamiento. Aunque OWL-S también incluye una parte de la descripción denominada *enlace al servicio* donde se define el mecanismo de acceso remoto al servicio, nosotros podemos obviarla ya que actualmente todos los métodos de jCOLIBRI 2 son locales al sistema y no se ejecutan remotamente.

En general, para describir un componente software se necesita especificar cuáles son sus precondiciones y postcondiciones. Según OWL-S estas condiciones son restricciones lógicas que deben cumplirse antes de ejecutar

el método (precondición) y que especifican el estado del mundo después de su ejecución (postcondición). Aunque estas restricciones pueden definirse en función de las entradas y salidas, se representan de forma independiente a dichos parámetros. Esto se debe a que en OWL-S las entradas y salidas únicamente especifican la transformación de datos producida por el servicio. Sin embargo, esta separación entre pre-/postcondiciones y entrada/salida no se realiza en WSMO. La razón es que también podemos ver las entradas y salidas como una serie de restricciones que deben cumplir los componentes para ser ejecutados. Por ejemplo, si un componente recibe un parámetro determinado, no podrá ejecutarse hasta que un componente anterior haya devuelto un valor que sea compatible.

Esta compatibilidad podemos verla simplemente a nivel de la jerarquía de tipos del lenguaje de programación utilizado. En Java diríamos que sólo podemos ejecutar un método que recibe un parámetro de tipo *Number* si algún método anterior ha devuelto un valor con ese tipo o del tipo de alguna subclase como *Integer*, *Double*, etc.

En jCOLIBRI hemos optado por la visión de WSMO sobre los requisitos de ejecución de los métodos, tomando las entradas y salidas como pre-/postcondiciones. Aunque esto parezca limitar la comprobación de requisitos, no es así por dos motivos. El primero es que seguimos manteniendo el concepto de pre-/postcondiciones que se utilizará a la hora de la composición y que además podría extenderse para definir futuras restricciones lógicas. El segundo y más importante, es que a la hora de comprobar la compatibilidad entre parámetros de entrada y salida no utilizamos la jerarquía de tipos Java sino nuestra propia jerarquía de tipos definida en CBROnto.

De esta forma, al especificar las entradas y salidas de un método de jCOLIBRI se utilizan los tipos de nuestra propia jerarquía de conceptos. Esta jerarquía puede extenderse fácilmente creando subconceptos que permitan realizar comprobaciones semánticas adicionales. A continuación utilizaremos un sencillo ejemplo para comprender el significado y ventajas en la utilización de una jerarquía conceptual del tipos.

Ejemplo

En los sistemas CBR tenemos normalmente un método que se encarga de cargar la base de casos. Este método devuelve un tipo Java `CBRCaseBase`. Informalmente su declaración Java sería:

```
CBRCaseBase getCaseBase()
```

Ahora pensemos la situación en que existiera una versión del método anterior que devolviera también un objecto `CBRCaseBase` pero que en vez de ser casos normales fueran casos textuales (aquellos casos que tuvieran algún atributo conteniendo texto). Desde el punto de vista de la sintaxis Java su declaración sería similar:

```
CBRCaseBase getTextualCaseBase()
```

Aunque se podría pensar que este método debiera devolver un valor con tipo `TextualCBRCaseBase` que extendiera `CBRCaseBase`, ésto no tendría sentido ya que su funcionalidad sería la misma puesto que los métodos para obtener los casos –tengan o no atributos textuales– serían exactamente iguales.

Sin embargo, al definir la semántica de estos dos métodos (proceso que se detallará posteriormente) podríamos especificar que:

```
getCaseBase() devuelve cCBRCaseBase
getTextualCaseBase() devuelve cTextualCBRCaseBase
```

Tanto `cCBRCaseBase` como `cTextualCBRCaseBase` serían conceptos pertenecientes a la jerarquía de tipos de CBROnto[2]. El primer concepto sería superconcepto del segundo. De esta forma, podríamos definir diferentes métodos que manipularan la base de casos y que teniendo la misma declaración a nivel Java fueran distintos a nivel semántico:

```
manipulaCaseBase(CBRCaseBase casebase)
manipulaTextualCaseBase(CBRCaseBase casebase)
```

Estos dos métodos podrían describirse semánticamente como:

```
manipulaCaseBase() tieneInput cCBRCaseBase
manipulaTextualCaseBase() tieneInput cTextualCBRCaseBase
```

De esta manera y utilizando un razonador de Lógicas Descriptivas podríamos comprobar que el método `manipulaCaseBase()` es compatible tanto con `getCaseBase()` y `getTextualCaseBase()` ya que la entrada `cCBRCaseBase` es un superconcepto de `cTextualCBRCaseBase`. Sin embargo, el método `manipulaTextualCaseBase()` sólo sería compatible con `getTextualCaseBase()`.

La jerarquía de tipos definida en CBROnto contiene y extiende la jerarquía de tipos de Java (jerarquía Java $\subseteq$ jerarquía CBROnto). Esto nos permite definir restricciones lógicas sobre las pre-/postcondiciones de un método que no podrían expresarse sin las capacidades lógicas de las ontologías. Para ello, simplemente hay que extender uno de los conceptos de CBROnto con un subconcepto que añada las restricciones deseadas. Generalmente las restricciones se añadirán extendiendo algún concepto que represente un tipo Java. Ese era el caso del ejemplo anterior donde simplemente se añadiría una restricción al concepto `cTextualCBRCaseBase` que indicara que todo caso contenido en

[2]Para facilitar la explicación añadiremos una "c" minúscula al principio cada identificador de un concepto y así diferenciarlo de las clases Java con el mismo nombre. Sin embargo, este prefijo no se añade realmente en CBROnto y no aparecerá en los próximos ejemplos de las siguientes secciones

él debe tener algún atributo de tipo textual. Sin embargo, también sería posible añadir restricciones sobre conceptos que ya añaden restricciones. Por ejemplo, podríamos definir un subconcepto `cPureTextualCBRCaseBase` que extendiera `cTextualCBRCaseBase` indicando que todos los atributos deben ser de tipo textual.

Como se puede observar en el ejemplo anterior, la especificación de las entradas y salidas de los métodos actúa también como precondición y postcondición de los mismos, no siendo necesario buscar un formalismo que permita expresar y razonar con estas restricciones lógicas de forma independiente a las entradas y salidas. De hecho, la elección de un formalismo que represente las pre-/postcondiciones es un problema con distintas soluciones dependiendo de las necesidades concretas de la aplicación y en OWL-S se opta por dejarlo abierto, ofreciendo la posibilidad de elegir distintos lenguajes lógicos.

Otra cuestión a plantear sobre la definición de las condiciones de ejecución de un componente es si éstas deben ser obligatorias o recomendables. En algunas situaciones dichas restricciones deben cumplirse obligatoriamente a la hora de ejecutar el componente, pero en otras podrían tomarse como una serie de preferencias opcionales que indican su idoneidad. En nuestro dominio, encontramos un claro ejemplo en los algoritmos de selección de casos por diversidad. Algunos de ellos requieren mucho tiempo de cálculo y por lo tanto se recomienda ejecutarlos con bases de casos pequeñas. Esto no impediría utilizarlos cuando la base de casos fuera grande ya que seguirían funcionando correctamente aunque tendrían un alto coste de tiempo de ejecución.

El lenguaje OWL-S no diferencia entre estos dos tipos de restricciones fuertes y débiles. Sin embargo, en WSMO sí que encontramos esta distinción al tener *pre-/postcondiciones* obligatorias que están diferenciadas de las *suposiciones/efectos*. Estos dos últimos componentes no se comprueban a la hora de componer el servicio pero deben cumplirse para garantizar el correcto funcionamiento del mismo. De esta forma, las suposiciones/efectos en WSMO representan restricciones débiles que el servicio tendría que cumplir para ser ejecutado satisfactoriamente.

En nuestro caso varios métodos de jCOLIBRI incluyen restricciones débiles que deberían tenerse en cuenta a la hora de componer los métodos. La aproximación elegida consiste en utilizar las preferencias opcionales para indicar al usuario que quizás ese método no es el más recomendable a la hora de resolver la tarea. Sin embargo, el usuario sería siempre libre de elegirlo aunque no se garantice un funcionamiento adecuado del método.

La representación de las restricciones débiles se realiza de una forma sencilla: al definir el tipo de un parámetro de entrada se puede especificar un tipo obligatorio (restricción fuerte) y un tipo recomendable (restricción débil). Para que el método sea ejecutable, el tipo obligatorio debe ser siempre

compatible con la salida de un método anterior, mientras que si no se cumple dicha restricción únicamente se indicará al usuario sobre este hecho.

Para entender el funcionamiento de las restricciones débiles utilizaremos un ejemplo sencillo basándonos en el tamaño de la lista de casos manejada. En la ontología tendríamos la siguiente jerarquía:

```
cRetrievalResultCollection
 |- cLargeRetrievalResultCollection
     |- cMediumRetrievalResultCollection
         |- cSmallRetrievalResultCollection
```

Ahora imaginemos dos métodos de selección de casos con la siguiente definición semántica:

```
select1() tieneInput cRetrievalResultCollection
select2() tieneInput  cRetrievalResultCollection
          prefiereInput cMediumRetrievalResultCollection
```

En el caso del primer método, al no especificarse ninguna preferencia se asume que ésta es igual al tipo obligatorio indicado (por lo tanto su preferencia sería `cRetrievalResultCollection`). El tipo obligatorio para el segundo método es también `cRetrievalResultCollection` pero prefiere el una colección media (`cMediumRetrievalResultCollection`). Si asignáramos a esta entrada un valor de tipo `cLargeRetrievalResultCollection` ambos métodos serían compatibles respecto a las restricciones fuertes ya que `cLargeRetrievalResultCollection` es un subconcepto de `cRetrievalResultCollection`. Sin embargo sólo el primero cumpliría sus restricciones débiles ya que `cLargeRetrievalResultCollection` no es subconcepto de `cMediumRetrievalResultCollection`.

6.3.2.2 Representación de las composiciones de métodos

Una vez expuesta la representación de los métodos como entidades independientes podemos pasar a presentar su la representación de las composiciones de las que forman parte. Cuando se componen ciertos métodos siguiendo un flujo de ejecución determinado necesitamos añadir información extra que represente el enlace entre los mismos. Esta información sobre la composición se basa en dos elementos distintos:

- Por un lado tendríamos el *flujo de ejecución* de los métodos en base a las secuencias, condicionales y bucles que conforman una composición determinada. En el caso de jCOLIBRI esta información se encuentra representada en las plantillas, y –como se explicó en la sección anterior– se basa en las primitivas de los servicios compuestos de OWL-S (secuencias, bucles, condicionales, ...).

- El otro elemento a tener en cuenta es el uso de variables para enlazar las salidas con las entradas de los distintos métodos. Estas variables intermedias se encuentran tanto en OWL-S como en WSMO y permiten definir el *flujo de datos* –no de ejecución– entre los métodos.

Las variables intermedias sirven para almacenar los valores devueltos por distintos métodos y que pudieran servir de entrada a métodos ejecutados posteriormente. Además, pueden utilizarse para indicar información extra que añada restricciones a la hora de realizar una composición de métodos determinada. Estas restricciones podrán establecerse por el diseñador eligiendo las diferentes subclases de la jerarquía de tipos en CBROnto.

Una vez que se ha realizado la composición de los métodos, podemos ver dicha composición como un único componente que tendrá sus propias entradas y salidas. Estas entradas y salidas serán después utilizadas por los distintos métodos internos, pero a nivel de composición son conceptos distintos. Por lo tanto, a la hora de instanciar una plantilla, también se deberán definir sus entradas y salidas a nivel de composición. Las variables de entrada deberán ser definidas por el usuario y pueden considerarse como parámetros de configuración del sistema generado.

En la herramienta de composición de métodos de jCOLIBRI 2 se utilizarán las variables intermedias para definir el flujo de datos entre métodos y añadir información extra. Además, se podrán definir las entradas y salidas globales de la composición. Todo este proceso se detallará en la Sección 6.5, donde se expondrá el funcionamiento de dicha herramienta.

Por último, hay que destacar que las variables intermedias conforman en cierto modo ese mismo objeto *Contexto* utilizado en jCOLIBRI 1 para componer los PSMs. En el Capítulo 3 se detallaba este mecanismo de composición explicando los inconvenientes encontrados a la hora de definir la entrada y salida de los métodos en base este objeto. El contexto suponía un mecanismo de pizarra donde cada método dejaba los datos producidos y buscaba los datos necesarios para su ejecución. Sin embargo, el mecanismo de pizarra presentaba varios problemas –principalmente de escalabilidad– y suponía una limitación a la hora de definir la signatura de los métodos del armazón. Por ello, la parte de jCOLIBRI 2 orientada a desarrolladores no utiliza el contexto para definir las signaturas de los métodos y se indican explícitamente las entradas y salidas necesarias.

6.3.2.3 Representación de los métodos en CBROnto

Teniendo en consideración los diferentes aspectos involucrados en la representación de los métodos, detallaremos cómo se ha realizado dicha representación en CBROnto. Los conceptos principales involucrados se muestran en

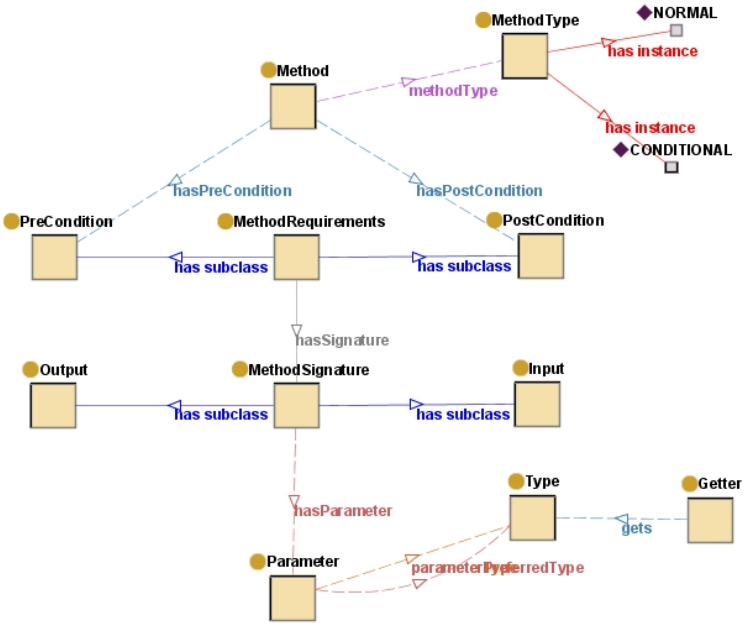


Figura 6.11: Esquema de la representación OWL de los métodos de jCOLIBRI2

la parte central de la Figura 6.8 y se detallan en la Figura 6.11. En esta representación, el concepto más importante es `Method`, al que pertenecerán las distintas instancias que representen cada método del armazón. Un método tiene un tipo: `Normal` o `Conditional` para indicar si el método resuelve una tarea simple normal o una tarea condicional (por lo que devolverá un valor booleano). Además, el método tendrá una serie de requisitos definidos en su precondición y postcondición. La pre-/postcondición tendrá una signatura concreta definida por sus entradas y salidas, las cuales estarán formadas por los diferentes parámetros del método. Por último, cada parámetro tendrá un tipo obligatorio y un tipo preferido para expresar las restricciones fuertes y débiles. En el Anexo B.3 se incluye el listado completo del código OWL de esta parte de CBROnto.

Un elemento que no había aparecido hasta ahora son los métodos *accesores*. En numerosas ocasiones se utilizan objetos Java para almacenar y encapsular la salida de un método, sobre todo si esta salida está compuesta por varios parámetros. Cuando se necesita acceder a dichos parámetros contenidos en el objeto se utilizan métodos accesores al estilo de los Java Beans presentados en la Sección 2.3.3. Por ejemplo, la lista de casos se almacena en un objeto `CBRCaseBase`, el cual dispone de un método `getCases()` para acceder a dicha lista que es de tipo `CBRCaseCollection`. Los distintos métodos de recuperación suelen tener como parámetro de entrada una lista

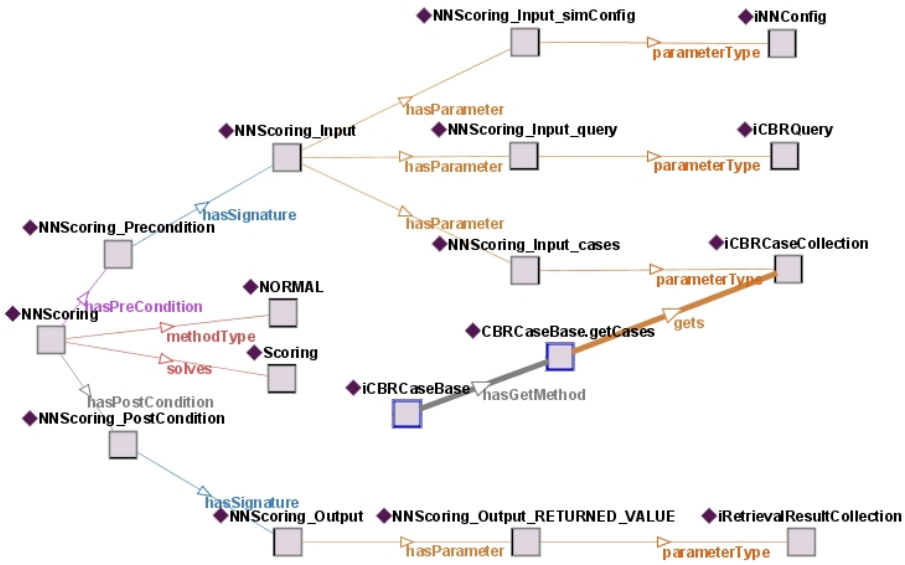


Figura 6.12: Representación OWL del método NNScoring

de casos de tipo `CBRCaseCollection`, pero el valor que se almacena en el contexto de la aplicación es de tipo `CBRCaseBase`. Por lo tanto, no sería posible realizar la composición semiautomática de dicho método ya que ambos tipos no serían compatibles. Sin embargo, invocando el método `getCases()` de `CBRCaseBase` sí que sería posible realizar dicha invocación. Este ejemplo pone de manifiesto la necesidad de conocer los métodos accesores de los objetos a la hora de realizar su composición. Por ello, en CBROnto se dispone de un concepto `Getter` que los representa y especifica el tipo devuelto por una método accesor concreto.

La Figura 6.12 muestra la representación mediante CBROnto del método de recuperación `NNScoring` (algunos elementos y relaciones se han omitido para facilitar la legibilidad). Como puede observarse, este método tiene tres entradas: consulta (de tipo `iCBRQuery`), configuración del método (`iNNConfig`) y lista de casos (`iCaseCollection`). La figura también muestra que este último tipo puede obtenerse invocando el método `getCases` del tipo `iCBRCaseBase`. De esto modo, es posible utilizar una variable de tipo `CBRCaseBase` para invocar el método `NNScoring` ya que la herramienta de composición puede inferir que hay un método accesor compatible gracias a la información de CBROnto. El siguiente listado ilustra esta misma situación mediante código fuente:

Listado 6.1: Ejemplo de utilización de los métodos accesores

```
//Signatura real del método
Collection<RetrievalResult> NNScoring(Collection<CBRCase> cases,
```

```
    CBRQuery query, NNConfig simConfig);

//Variables a utilizar
CBRCaseBase caseBase;
CBRQuery query;
NNConfig simCofig;
Collection<RetrievalResult> retrievedSet;

//Asignación incorrecta
retrievedSet = NNScoring(caseBase, query, simConfig);

//Asignación correcta usando métodos accesores
retrievedSet = NNScorint(caseBase.getCases(), query, simConfig);
```

6.3.3 El enlace entre las dos capas de jCOLIBRI 2

Una vez que hemos estudiado la representación semántica de los métodos en CBROnto, llegamos a un punto muy importante en la plataforma que consiste en obtener dicha representación automáticamente a partir del código fuente. Como se observa en el esquema principal de la arquitectura de jCOLIBRI 2 mostrado en la Figura 3.11 (pág. 94), existe una capa orientada a desarrolladores y otra orientada a diseñadores. Ambos trabajan sobre los mismos métodos: los primeros utilizando el código directamente y los segundos a través de la herramienta de composición. Es aquí donde surge la cuestión de cómo enlazar el código fuente de los métodos con su representación semántica para permitir que la herramienta de composición conozca su funcionamiento.

La respuesta a estas cuestiones la encontramos en las *Anotaciones*. Las anotaciones representan meta-información sobre el código fuente y permiten documentar tanto clases como métodos o variables. Un ejemplo de anotación serán las etiquetas *JavaDoc* que permiten documentar el funcionamiento del código fuente. Ejecutando una sencilla herramienta podemos obtener dichas anotaciones y generar el código html con la documentación. Aunque las anotaciones *JavaDoc* han venido incorporadas en la plataforma Java desde sus inicios, es a partir de la versión 1.5 cuando se amplia este mecanismo para permitir que los desarrolladores puedan definir y manipular sus propias anotaciones.

Gracias a las anotaciones, en jCOLIBRI 2 es posible anotar cada método con su información semántica y permitir su análisis automático por la herramienta de composición. El uso de las anotaciones presenta numerosas ventajas:

- No es necesario modificar en absoluto el código funcional del método. Simplemente se añadirá una anotación en cada uno que no interferirá

con sus sintaxis y declaración Java.

- Se evita gestionar un fichero aparte con la información semántica. Antes de la aparición de las anotaciones, las aplicaciones solían utilizar un fichero aparte con la meta-información requerida. Esto provocaba distintos problemas al tener que mantener dos fuentes de información separadas pero ligadas entre ellas. Este mismo problema ya se detalló en la Sección 3.1.3 donde se exponían los inconvenientes encontrados en la versión inicial de jCOLIBRI donde se manipulaban diversos ficheros XML de configuración con información sobre la estructura del caso, funciones de similitud, etc.

- Las anotaciones pueden manipularse en tiempo de ejecución, lo que permite que nuestra herramienta busque los métodos anotados automáticamente.

- El desarrollador es completamente libre de anotar o no sus métodos. Si un método no estuviera anotado con la información semántica simplemente no aparecerá como disponible en la herramienta de composición de jCOLIBRI 2, pero ésto no influirá en su utilización por otros desarrolladores.

- Las anotaciones sirven también como documentación sobre el funcionamiento de los métodos y aparecerán en el API html generada mediante la herramienta JavaDoc.

Las anotaciones para representar la información semántica de los métodos de jCOLIBRI 2 se muestran en el siguiente listado y se describen a continuación:

Listado 6.2: Definición de la anotación de los métodos

```
public @interface JCOLIBRIMethod {

    public enum MethodType  {NORMAL, CONDITIONAL};

    String solvesTask();
    String alias();
    MethodType methodType();
    String description();
    Parameter[] input();
    Parameter[] output();
}

public @interface Parameter {
    String name() default "";
    String type();
    String preferredType() default "";
    String description();
```

```
}

public @interface JCOLIBRIGetter {
    String source();
    String target();
    String description();
}
```

Como puede observarse, las anotaciones Java se definen de forma similar a las interfaces. La anotación principal es `JCOLIBRIMethod` que se compone de la tarea resuelta, el "alias" del método (nombre que se utilizará para identificarlo ya que su nombre real puede ser poco descriptivo), el tipo de método (normal o condicional), una descripción, y los parámetros de entrada y salida.

Los parámetros se definen con otra anotación distinta denominada `Parameter` ya que una anotación puede estar compuesta de otras anotaciones –es decir, que es posible anotar anotaciones. Cada parámetro tendrá un nombre, un tipo, un tipo preferido y una descripción. Algunos de estos valores –como el nombre o el tipo preferido– pueden ser optativos y por eso tienen un valor por defecto vacío. La razón de tener parámetros que no tengan nombre es que los valores devueltos por las funciones son parámetros sin nombre. Además, la gran mayoría de métodos no definirán un valor de tipo preferido como restricción débil. Como ya se explicó en la sección anterior, en ese caso se tomará el tipo obligatorio como tipo preferido.

Por último en el listado aparece la anotación utilizada para identificar los métodos accesores. Esta se compone simplemente del tipo fuente, destino y la descripción del método accesor.

Para acabar de entender el mecanismo de las anotaciones, veamos un ejemplo real de anotación:

Listado 6.3: Ejemplo de método anotado

```
@JCOLIBRIMethod(
 solvesTask="SelectCases",
 alias="GreedySelection",
 methodType=MethodType.NORMAL,
 description="This␣method␣incrementally␣builds␣a...",
 input = {
  @Parameter(name="cases",
             type="RetrievalResultCollection",
             preferredType="SmallRetrievalResultCollection",
             description="Cases␣list␣where␣select"),
  @Parameter(name="query",
             type="CBRQuery",
             description="Query␣used␣to␣compute␣diversity"),
  @Parameter(name="simConfig",
```

```
            type="NNConfig",
            description="NN␣configuration␣object"),
  @Parameter(name="k",
            type="Integer",
            description="Number␣of␣cases␣to␣select"),
 },
 output = {
  @Parameter( type="CBRCaseCollection",
              description="Selected␣cases")
 }
)
public static Collection<CBRCase> greedySelection(Collection<
    RetrievalResult> cases, CBRQuery query, NNConfig simConfig, int k)
    {
  ...
}
```

El ejemplo muestra la anotación del método `GreedySelection`. La declaración Java normal del método aparece en la parte inferior del listado, donde podemos observar que recibe una colección de casos recuperados, una consulta, la configuración de la similitud del método del vecino más próximo, y el número de casos a seleccionar. La salida del método será una colección de casos (`CBRCaseCollection`).

La anotación de este método empieza con la tarea que es capaz de resolver (`SelectCases`), su alias, tipo y descripción. La tarea indicada deberá ser una de las tareas definidas en CBROnto. A continuación se definen los parámetros de entrada con su nombre, tipo y descripción. En el caso del primer parámetro –`cases`– también se indica un tipo preferido ya que este método en concreto no funciona bien con bases de casos grandes. Al igual que las tareas resueltas, los tipos deberán estar definidos en la ontología. Finalmente se define el parámetro de salida que carece de nombre al ser el valor devuelto por el método.

Respecto a los métodos accesores, el siguiente listado muestra un ejemplo de uso real:

Listado 6.4: Definición de la anotación de los métodos accesores

```
@JCOLIBRIGetter(
 source="CBRCaseBase",
 target="CBRCaseCollection",
 description="Returns␣loaded␣cases"
)
public Collection<CBRCase> getCases(){
  ...
}
```

Figura 6.13: Código OWL representando un método de jCOLIBRI2

La interpretación de esta anotación es muy sencilla ya que sólo es necesario especificar el objeto que define el método accesor (`source`), el tipo devuelto (`target`) y una descripción de su funcionamiento. A partir de la anotación anterior se obtendría automáticamente la representación OWL que se muestra de forma resaltada en la Figura 6.12.

En la anotación de los métodos se utilizan conceptos que deben estar definidos en la ontología y que se utilizarán al generar la representación OWL. Aquí aparece una pregunta obvia: ¿qué ocurriría si el desarrollador quisiera utilizar un tipo (o incluso tarea) que no estuviera predefinida en CBROnto? La solución a este problema es tecnológicamente sencilla ya que el estándar OWL incluye un mecanismo por el que se pueden extender fácilmente una

ontología simplemente importándola en otra. De esta forma, el desarrollador podría definir su propia extensión de CBROnto creando los conceptos que necesitase en función de los ya definidos en nuestra ontología. Esta ontología extendida sería la utilizada por la herramienta de composición en vez de usar la ontología CBROnto original.

Habiendo definido el mecanismo de anotación semántica de los métodos, la generación de su representación OWL como instancias de los conceptos mostrados en la Figura 6.11 se realiza de forma automática. Al iniciar la herramienta de composición –que se detallará en la Sección 6.5– se analizan todas las clases del armazón y se buscan los métodos que contengan las anotaciones. Hay que recordar que este análisis es posible gracias a que las anotaciones se mantienen en el código una vez compilado. Por cada método encontrado se generaran las distintas instancias que lo representan, enlazadas a través de las propiedades definidas en CBROnto. La Figura 6.13 muestra una representación del código OWL generado a partir de la anotación mostrada en el Listado 6.3. En esta figura hay que destacar la propiedad `methodRealPath` que no aparecía en las figuras anteriores al ser una propiedad de datos (String) y que indica el nombre o camino real del método en el código para que pueda ser utilizado en la etapa de adaptación para generar una aplicación CBR que lo invoque. En el Anexo B.4 se incluye el listado completo del código OWL generado por la herramienta para este método.

La utilización de los estándares de los Servicios Web Semánticos añaden la posibilidad de poder utilizar las herramientas de composición de métodos de forma distribuida. De esta forma se permitiría la composición de métodos remotos (servicios) que estuvieran ejecutándose en una máquina distinta. Esta característica sería transparente al usuario y permitiría mantener la aproximación aquí expuesta donde se sigue un proceso CBR para la composición de los sistemas CBR ((Recio-García et al., 2008b, 2005b).

Una vez descritos los mecanismos de representación de las plantillas y los métodos, podemos pasar a detallar nuestra herramienta para el diseño de sistemas CBR siguiendo esa misma filosofía CBR: recuperar casos (plantillas) similares al sistema deseado y adaptarlo a los requisitos del usuario. Estas dos partes de la herramienta se presentan en las dos próximas secciones. La Sección 6.4 explicará el mecanismo de recuperación y la Sección 6.5 presenta el proceso de adaptación.

6.4 Recuperación de plantillas

El proceso de diseño de sistemas CBR mediante plantillas empieza por la recuperación de la plantilla más adecuada para el sistema a desarrollar. La

idea original consistía en definir un lenguaje que sirviera para describir cada plantilla desde el punto de vista de sus requisitos funcionales y no funcionales. Ese mismo lenguaje podría usarse por el diseñador del sistema CBR para construir una consulta con los requisitos necesarios de su aplicación. Entonces, un motor de recuperación podría encontrar la plantilla más adecuada para esos requisitos.

Sin embargo, pronto se llegó a la conclusión de que esa aproximación presentaba dos grandes problemas. El primero radica en las dificultades que encontraría el diseñador a la hora de expresar sus requisitos mediante ese lenguaje. El segundo inconveniente es que las plantillas por sí mismas no son lo suficientemente descriptivas para el usuario. Esto se debe a que normalmente lo más que se puede decir de una plantilla es que las tareas genéricas se descomponen de cierta manera en tareas simples. Esta descripción deja de lado los métodos asignados para resolver las tareas, y esta característica es también muy importante a la hora de describir las plantillas. Dicho de otra manera, la conclusión es que las descripciones necesarias en la recuperación deben incluir aspectos de los sistemas implementados a partir de las plantillas, no de las plantillas independientemente.

Esta conclusión llevó a la definición de un sistema recomendador de plantillas basado en casos que se describe a lo largo de esta sección ya que se evita el uso de una consulta. Es decir, la recuperación de plantillas de recomendadores se realiza mediante un sistema de recomendación. Este sistema de recomendación de las plantillas se construyó además utilizando una de las plantillas que recomienda, lo que valida en cierto modo el mecanismo de composición presentado en este capítulo.

A lo largo de esta sección se presentará el recomendador de plantillas basado en casos (Secciones 6.4.1 y 6.4.2) junto con una descripción detallada de las medidas de similitud utilizadas por este sistema ya que son el elemento más importante del mismo (Sección 6.4.3). En la Sección 6.4.4 se describe un ejemplo práctico que permite entender el mecanismo de recuperación y las medidas de similitud utilizadas. Por último, la Sección 6.4.5 incluye una evaluación empírica de dichas medidas de similitud que se realizó para encontrar la mejor ponderación de los factores que influyen en la misma. Los resultados expuestos en esta sección se encuentran publicados en Recio-García et al. (2008a).

6.4.1 Recomendación de plantillas basada en casos

Si la recuperación de las plantillas se realiza mediante un recomendador basado en casos, debemos estudiar esta etapa desde el punto de vista de la base de casos de dicha aplicación –que se compondrá de plantillas– y del comportamiento del recomendador implementado. Como ya se señalaba en la introducción de esta sección, las plantillas no son suficientes a la hora de realizar la recomendación y también es necesario incluir información sobre

los sistemas potencialmente implementables a partir de ellas. Esta estructura de la base de casos es la que presentaremos en la Sección 6.4.1.1. Respecto al funcionamiento del recomendador, el mecanismo elegido supone una novedad en el campo de la recomendación que hemos denominado *"Recuperar probando"* (Retrieval-by-Trying). Este mecanismo se detalla en la próxima Sección 6.4.2.

6.4.1.1 Los casos

Los casos de nuestro recomendador de plantillas son sistemas CBR que sirvan como referencia en la implementación de nuevas aplicaciones. Por lo tanto, utilizando la nomenclatura definida en la Sección 6.2 diríamos que los casos son sistemas o plantillas ejecutables. Es decir, plantillas a cuyas tareas se les ha asignado un método capaz de resolverlas. Sin embargo, las plantillas a partir de las cuales se han generado los sistemas también se almacenan en el caso. Podría decirse que el sistema es la *descripción* del caso y la plantilla desde la que se generó es la *solución* del mismo.

En algunas situaciones, el sistema en la base de casos podrá ser una aplicación original que haya sido desarrollada en los últimos años. En otras situaciones esto puede que no sea posible, lo que limitaría la variedad y completitud del sistema. Entre los ejemplos de estas aplicaciones originales cuya implementación no es accesible y no pueden ser incluidas directamente en la base de casos podríamos encontrar los sistemas Entree de Burke et al. (1997b) y ExpertClerk de Shimazu (2002). Para solucionar este problema y poder ofrecer una base de casos completa se re-implementaron versiones simplificadas de estos sistemas y se añadieron a la base de casos junto con sus plantillas. Esta implementación de sistemas tan importantes en la literatura de investigación y desarrollo en recomendadores basados en casos sirve de testimonio sobre las capacidades ofrecidas en jCOLIBRI tanto desde el punto de vista de las plantillas como de los métodos, ya que dichas implementaciones se realizaron de una manera bastante sencilla y directa.

La base de casos sobre la que se realizó el trabajo expuesto en este capítulo contenía 14 sistemas finales totalmente implementados, aunque la inclusión de otros sistemas podría realizarse muy fácilmente. A continuación detallaremos el recomendador implementado para guiar al usuario en la recuperación de estos sistemas.

6.4.2 Recuperar probando

Como ya se ha comentado anteriormente, el proceso de recuperación de plantillas presenta dos problemas principales. El primero consiste en que la descripciones necesarias a la hora de realizar las recomendaciones deben incluir características de los sistemas implementados a partir de las plantillas y no

sólo características directas de las mismas. Este problema se resuelve incluyendo dichos sistemas en la base de casos, tal y como se presentaba en la sección anterior. El segundo problema es que el diseñador podría encontrar dificultades para expresar los requisitos de su sistema a la hora de crear una consulta.

Este último problema no aparece únicamente en el ámbito de la recomendación de plantillas basada en casos. En otros dominios de sistemas de recomendación es común que no sea fácil para los usuarios realizar la descripción de sus preferencias. Sin embargo, la investigación en sistemas de recomendación ofrece una solución a este problema.

En algunos de los sistemas de recomendación más importantes e influyentes, los usuarios pueden expresar sus preferencias mediante alguna de las formas de navegación basada en propuestas (Burke et al., 1997b; McGinty y Smyth, 2002; Shimazu, 2002; Smyth y McGinty, 2003). Este tipo de navegación se detalló en la Sección 6.2.1.3 y consistía en presentar al usuario un pequeño conjunto de artículos a partir del cual se seleccionaba el más similar al artículo deseado. El siguiente conjunto propuesto al usuario contendría los artículos más parecidos al seleccionado, teniendo en cuenta otro tipo de información aportada por el usuario (normalmente en forma de críticas).

Esta es la idea que se sigue en nuestro recomendador de plantillas basado en casos. En él se muestran dos sistemas al usuario –que es el diseñador de aplicaciones CBR. Siguiendo la aproximación de hacer recomendaciones basadas en comparaciones de McGinty y Smyth (2002), el usuario elegirá uno de los sistemas propuestos y nuestro recomendador intentará recuperar más sistemas que sean similares al seleccionado. De esta forma se soluciona el problema de que el usuario no sea capaz de describir los requisitos del sistema que desea implementar.

Pero aquí aparece otro problema. En cualquier iteración, ¿cómo sabe el diseñador qué sistema elegir? Si la única información mostrada fuera el nombre del sistema o algún tipo de descripción sobre el mismo, sería muy difícil que el diseñador pudiera hacerse una idea clara sobre las opciones presentadas y no podría realizar su elección. Sin embargo, hay que tener en cuenta que el usuario está eligiendo entre sistemas (plantillas ejecutables) que ya están implementadas. Por lo tanto, lo que hace nuestro recomendador es permitir *ejecutar* dichos sistemas. Este proceso lo hemos bautizado como *Recuperar probando*: el usuario puede *probar* el artículo recomendado (en este caso los sistemas de recomendación que se están recomendando) para poder realizar su elección.

La Figura 6.14 muestra una captura de nuestro sistema de recuperación mediante esta aproximación donde puede observarse que el recomendador está proponiendo dos sistemas al usuario. Para realizar la elección, el usuario puede ejecutar ambos sistemas a través del botón *"Try it!"*.

La *recuperación-probando* ("Retrieval-by-Trying") es una aproximación

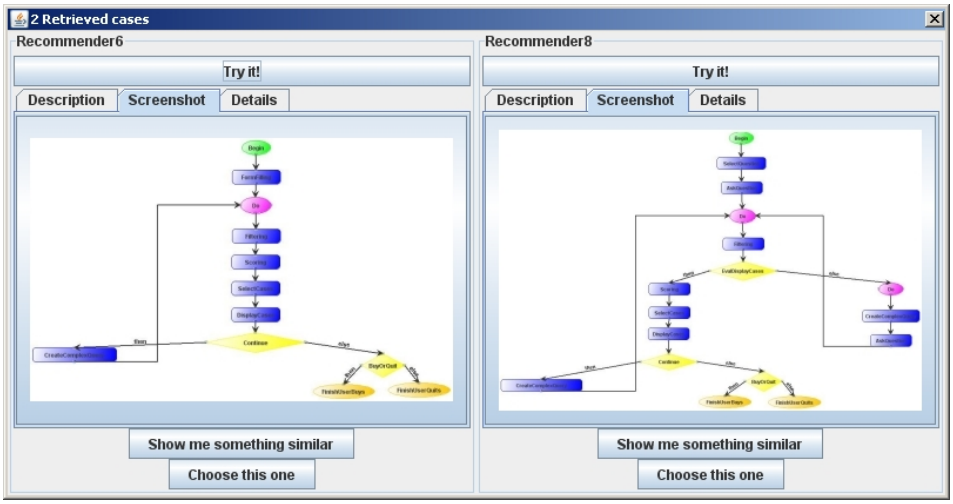


Figura 6.14: Captura de la herramienta de recuperación de plantillas

natural para sistemas que son relativamente simples e interactivos como los sistemas de recomendación basados en casos. Quizás este mecanismo no pudiera escalar bien para otros tipos de sistemas CBR que sean más complejos (p.e. planificadores basados en casos) o que no sean interactivos (p.e control de procesos). Sin embargo, en nuestra opinión sí que sería suficiente para el tipo de aplicaciones que pueden ser desarrolladas mediante nuestro armazón. Otra restricción de esta aproximación es que necesita una base de casos –o productos– previamente configurada.

Las próximas subsecciones explican los siguientes aspectos de nuestra aproximación: cómo se seleccionan los sistemas mostrados inicialmente al usuario, cómo se eligen los subsiguientes pares de sistemas, y cómo se maneja la información aportada por el usuario en cada iteración.

6.4.2.1 Puntos de entrada al sistema

Inicialmente, el recomendador selecciona dos sistemas que serán propuestos al usuario. Para ello se utiliza el método de recuperación basado en la media expuesto por Shimazu (2002) y que se incluye en jCOLIBRI.

Este método se describió en la Sección 6.2.2.1 y consiste en crear un caso artificial que represente la mediana de la base de casos. El primer sistema propuesto al usuario es el más cercano a este caso medio. Aunque en el algoritmo original se eligen otros dos artículos que sean lo más distintos posible a la media, en nuestro dominio hemos considerado que es mejor elegir sólo otro artículo debido a la complejidad de los casos. De cualquier modo, el número de opciones presentadas podría modificarse de forma sencilla.

6.4.2.2 Recuperación mejorada mediante diversidad para recomendaciones basadas en comparación

Al usuario se le muestran dos sistemas que debe probar. Al estilo de la navegación mediante propuestas, elegirá uno de ellos y nuestro recomendador recuperará un nuevo par de sistemas que sean similares al seleccionado por el usuario.

Para seleccionar los siguientes sistemas, necesitamos asegurarnos que además sean distintos entre ellos, ya que si fueran demasiado parecidos existen menos posibilidades de que alguno de ellos satisfaga los requisitos del usuario. Utilizando la terminología propia de los sistemas de recomendación necesitamos aumentar la *diversidad* del conjunto recuperado (Smyth y McClave, 2001). Para ello, se ha utilizado el conocido algoritmo de selección voraz limitado a un subconjunto aleatorio de artículos ("Bounded Greedy Selection") que se incluye en jCOLIBRI y que también se describió en la Sección 6.2.2.2.

6.4.2.3 Gestión de las preferencias del usuario

El caso seleccionado por el usuario nos ofrece información acerca de sus preferencias. Sin embargo, los casos rechazados también ofrecen información importante. En la Sección 6.2.2.4 se presentaban las estrategias planteadas en McGinty y Smyth (2002) para gestionar la mejora de las preferencias del usuario al utilizar navegación mediante propuestas. Estas estrategias se encuentran implementado en el armazón y se denominan: Más como esto (More Like This - MLT), Más como esto ponderado (weighted More Like This - wMLT), Más como esto parcial (partial More Like This - pMLT), Menos cómo esto (Less Like This - LLT) y Más-Menos como esto (More+Less Like This - M+LLT).

El recomendador de plantillas puede trabajar con cualquiera de ellas. Sin embargo, en la Sección 6.4.5 se presenta una evaluación empírica que permitió elegir la que ofrece mejores resultados en el dominio de la recomendación de plantillas.

6.4.3 Cálculo de la similitud para la recomendación de plantillas

La descripción de las plantillas se realiza mediante un conjunto de pares atributo-valor. Como se describirá a lo largo de esta sección, este conjunto de atributos representa las tareas del sistema, los métodos utilizados para resolver dichas tareas, y una serie de características semánticas definidas por el experto en el dominio y que se almacenan en CBROnto. Gracias a esta representación, podemos calcular la similitud basándonos en *qué* hace el sistema (comparando la estructura de tareas); *cómo* hace lo que hace (comparando sus métodos); y calculando su similitud de acuerdo con las características

globales definidas por el experto y que describen el comportamiento y estructura del sistema a alto nivel.

La utilización de las características semánticas se debe a la necesidad de incluir en la medida de similitud ciertos aspectos que no están reflejados en la estructura básica de tareas o la asignación de métodos. Un buen ejemplo de estos aspectos los encontramos en el dominio de los recomendadores. Las plantillas y sistemas que se han incluido en jCOLIBRI 2 pueden clasificarse de acuerdo al tipo de información solicitada al usuario: todas sus preferencias (formulario), algunas preferencias (navegación mediante preguntas), a través de un perfil, o utilizando recomendaciones pasadas. En esta clasificación podríamos definir el primer grupo de sistemas como "todos aquellos que tengan la tarea *FormFilling*", pero para definir los recomendadores que utilizan recomendaciones anteriores tendríamos que utilizar: "todos aquellos recomendadores que contengan la tarea *ReadProfile* y utilicen el método *CollaborativeRetrieval* para resolver la tarea *Retrieval*". Esta última definición pone de manifiesto la necesidad de mezclar características estructurales y funcionales para clasificar los sistemas. Obviamente, estas características deberán ser definidas por un experto en el dominio que tenga un conocimiento profundo en el funcionamiento del tipo de sistemas CBR en cuestión.

Nuestro algoritmo obtiene un valor final de similitud mediante una combinación ponderada de las tres aproximaciones anteriores similitud entre estructura de tareas, similitud entre métodos y similitud entre características globales. Esta medida se formaliza en la siguiente fórmula:

$$sim(t_1, t_2) = w_1 \cdot ts(t_1, t_2) + w_2 \cdot mc(t_1, t_2) + w_3 \cdot gf(t_1, t_2)$$

donde $ts()$ (task similarity) es la similitud de la estructura de tareas, $ms()$ (methods similarity) es la similitud entre los métodos asignados para resolver la misma tarea, y $gf()$ (global features) es la estimación de similitud de acuerdo a las características globales.

Los pesos óptimos w_1, w_2, w_3 se obtuvieron de forma empírica a través de una serie de experimentos que se describirán en la Sección 6.4.5. Antes, detallaremos el funcionamiento de cada una de las tres medidas de similitud utilizadas.

6.4.3.1 Similitud basada en la estructura de tareas

La comparación basada en la estructura de tareas se basa en que cada sistema es una instanciación de una plantilla final que a su vez es una descomposición de una plantilla genérica. Estas plantillas genéricas suelen tener en común varias tareas genéricas que habrán sido descompuestas en secuencias de tareas simples de distinta forma al generar las plantillas finales. De este modo, por cada tarea genérica común a dos plantillas podemos comparar la forma en que se descompuso en tareas simples.

Definimos $G = \{G_1, G_2, G_3, \ldots, G_n\}$ como el conjunto de tareas genéricas (mostradas en rojo en las Figuras 6.2, 6.3, 6.4 y 6.5), y $Q = \{Q_1, Q_2, Q_3, \ldots, Q_n\}$ como el conjunto de posibles descomposiciones en tareas simples de las tareas $G_i \in G$. Cada secuencia Q_i se compone de un conjunto de tareas simples $S = \{S_1, S_2, S_3, \ldots, S_n\}$. Las Figuras 6.6 y 6.7 muestran ejemplos de dichas descomposiciones. En la representación de las plantillas se define un par atributo-valor por cada tarea genérica en G. Los valores de dichos atributos serán las secuencias en tareas primitivas Q_i que descomponen cada una de las tareas genéricas. La similitud de dos plantillas de acuerdo a la estructura de tareas se realiza comparando estos atributos mediante una función de igualdad que devuelve 1 si coinciden las descomposiciones o 0 e.o.c. Finalmente se realiza una media entre los valores obtenidos.

6.4.3.2 Similitud basada en los métodos

El computo de la similitud basada en los métodos significa obtener un valor numérico que representa *cómo* se realizan las distintas tareas del sistema. Para ello, la representación de cada sistema incluye diferentes atributos, uno por cada tarea simple en la plantilla. Los valores permitidos para cada uno de estos atributos serán el conjunto de métodos capaces de resolver dicha tarea. Si una tarea simple aparece más de una vez en la plantilla se representa por atributos diferentes.

Para comparar los métodos se ha creado una serie de conceptos en CBR-Onto que representan y organizan jerárquicamente su comportamiento. De esta forma, existirá un individuo por cada método en el armazón que será asertada como instancia bajo el concepto más apropiado. Esta organización se muestra en la Figura 6.15 y permite comparar los métodos mediante una de las funciones de similitud con ontologías presentadas en el Capítulo 5 (en este caso hemos elegido la función coseno).

6.4.3.3 Similitud basada en características globales

En la medida de similitud basada en características globales se utiliza CBR-Onto para almacenar y representar el conocimiento del experto del dominio necesario para clasificar los sistemas. Como ya se comentó anteriormente, en ciertas ocasiones los sistemas deben ser clasificados de acuerdo a características heterogéneas definidas como combinaciones de las tareas y métodos utilizados (por ejemplo: "los recomendadores colaborativos son aquellos con la tarea *ReadProfile* y el método *CollaborativeRetrieval*". Estas características sólo pueden obtenerse de la experiencia del experto en el desarrollo de dichos sistemas y podrían considerarse como varios ejes de clasificación que ayudan a catalogarlos y organizarlos. Por ejemplo, en el dominio de los recomendadores podemos clasificar las aplicaciones dependiendo del tipo de adquisición de preferencias: navegación mediante preguntas o navegación

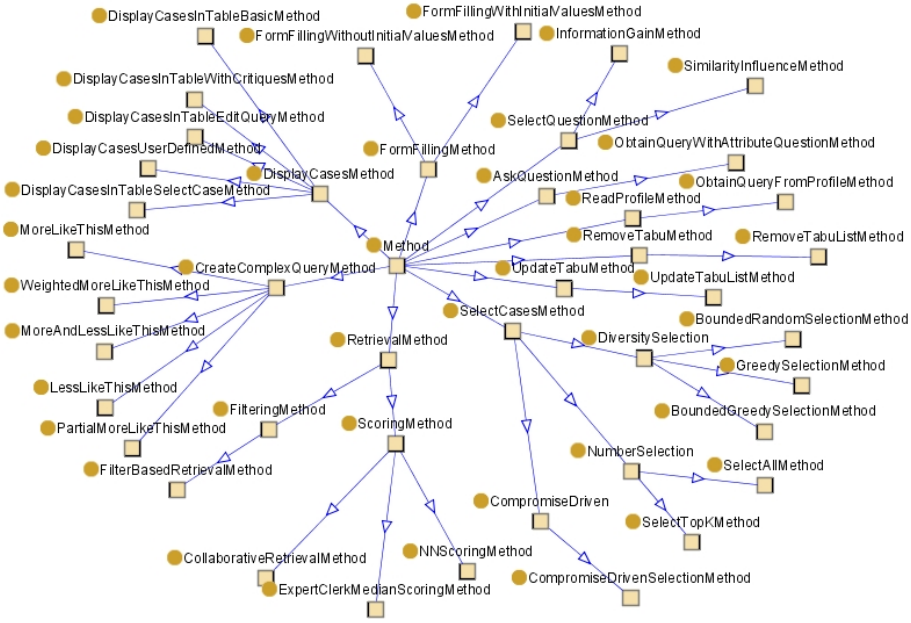


Figura 6.15: Organización jerárquica de los métodos en CBROnto

mediante propuestas. Otra clasificación posible sería el tipo de recuperación: mediante filtrado o mediante puntuación de los casos. Estas dos características (tipo de adquisición de preferencias y tipo de recuperación) definen dos puntos de vista distintos que permiten clasificar los recomendadores, y por extensión las plantillas que los generaron.

Es necesario destacar que estas características globales pueden solapar, es decir, una misma plantilla puede estar clasificada en varios ejes. Por ejemplo, las plantillas pueden clasificarse de acuerdo al tipo de información solicitada al usuario: todas las preferencias, algunas preferencias, mediante perfiles, etc. Sin embargo, esta clasificación es independiente del tipo de navegación y un sistema que utilice navegación mediante preguntas también será catalogado como sistema que solicita al usuario algunas preferencias.

Todas estas características (tipo de navegación, proceso de recuperación, o información solicitada al usuario) nos sirven para clasificar las plantillas desde diferentes puntos de vista y para ello las definimos como conceptos definidos de la ontología CBROnto. La parte izquierda de la Figura 6.16 muestra los distintos conceptos incluidos en CBROnto para representar las características globales que clasifican los sistemas de recomendación. Cada característica es un subconcepto directo del concepto *ImplementedSystem*. La primera de ellas es *ByCasesSelectionMethods* y clasifica los recomendadores de acuerdo al tipo de selección de casos utilizado: sin selección o con selección tanto por similitud directa como por diversidad. La siguiente característica

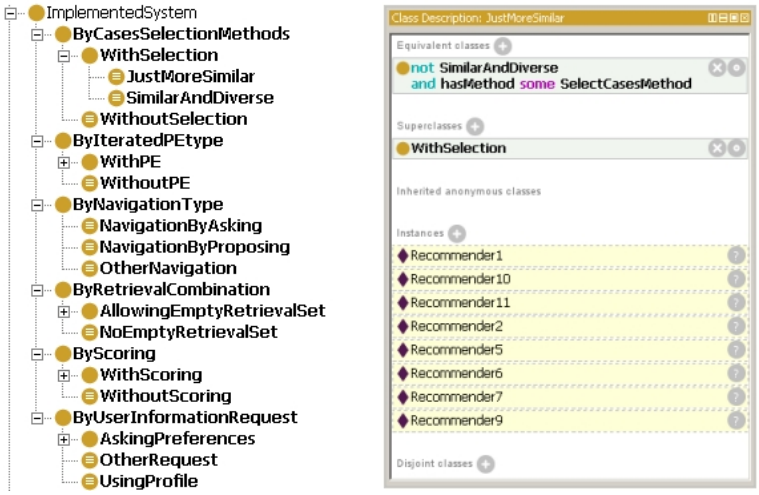


Figura 6.16: Clasificación de los sistemas de acuerdo a características globales

clasifica de acuerdo a la inclusión o no de la etapa de adquisición iterativa de preferencias. Siguiendo esta idea, el resto de conceptos definen diferentes ejes de clasificación: por el tipo de navegación, tipo de recuperación, tipo de puntuación utilizada y por el tipo de información solicitada al usuario.

Cada una de las características se describe mediante predicados de Lógica Descriptiva que representan una combinación particular de tareas y métodos. Por ejemplo, la parte derecha de la Figura 6.16 muestra la definición del concepto "JustMoreSimiliar" como aquellos sistemas que no pertenezcan a la categoría "SimilarAndDiverse" y que utilicen algún método perteneciente al concepto "SelectCasesMethod" (según la organización de métodos mostrada en la Figura 6.15). Además, cada sistema se representa en la ontología como una instancia que contiene una serie de propiedades que describen los métodos y tareas implicados. De esta forma, utilizando los razonadores de Lógicas Descriptivas podemos clasificar automáticamente las instancias que representan cada sistema en la jerarquía de características globales. Esta clasificación basada en el reconocimiento de instancias se muestra también en la Figura 6.16 (parte derecha) donde aparecen las instancias que han sido inferidas como pertenecientes al concepto "JustMoreSimilar".

Si cada característica global define su propio subárbol de conceptos, las instancias que representan a los sistemas serán reconocidas bajo alguno de los conceptos de cada subárbol. Es decir, una misma instancia aparecerá una vez en cada uno de los subárboles. Esto se debe al solapamiento de las características, que como ya se comentó, son independientes entre ellas.

Para obtener la similitud entre dos sistemas de acuerdo a las características globales, se obtiene la distancia de los dos individuos que los representan

		Median Case	Recommender 6	Recommender 8
Generic Template		Conversational A	ConversationalA	ConversationalB
Task Structure	One-Off P.E.	Form_Filling	Form_Filling	Ask_Question
	Retrieval	Filtering_Scoring_Selection	Filtering_Scoring_Selection	Filtering
	Iterated P.E.	Create_Complex_Query	Create_Complex_Query	ExpertClerck_Iterated_P.E.
	Display	Display	Display	ExpertClerck_Display
Methods	FormFilling	FormFillingWithInitialValues	FormFillingWithoutInitialValues	
	SelectQuestion	InformationGain		InformationGain
	AskQuestion	ObtainQueryWithAttributeQuestion		ObtainQueryWithAttributeQuestion
	ReadProfile	ObtainQueryFromProfile		
	Scoring	ExpertClerkMedianScoring		
	SelectCases	SelectTopK		
	DisplayCases	DisplayCasesTableWithCritiques		
	Filtering	FilterBasedRetrievalMethod	FilterBasedRetrievalMethod	FilterBasedRetrievalMethod
	Scoring	NNScoringMethod	NNScoringMethod	NNScoringMethod
	Selection	SelectTopK	SelectTopK	BoundedGreedySelection
	RemoveTabu	null		
	Display	DisplayCasesTableWithCritiques	DisplayCasesTableWithCritiques	DisplayCasesTableWithCritiques
	UpdateTabu			
	FormFilling	FormFillingWithInitialValues		
	SelectQuestion	InformationGain		InformationGain
	AskQuestion	ObtainQueryWithAttributeQuestion		ObtainQueryWithAttributeQuestion
	CreateComplexQuery	MoreLikeThis	MoreLikeThis	MoreLikeThis
Global Features	CasesSelectionMethods		JustMoreSimilar	SimilarAndDiverse
	IteratedPEtype		ModifyingQueryWithUserSelection	ModifyingQueryWithUserSelection
	NavigationType		NbP	NbP and NbA
	RetrievalCombination		NotEmptyRetrievalSet	NotEmptyRetrievalSet
	Scoring		BasicScoring	BasicScoring
	UserInformationRequest		AskingUserForAllPreferences	AskingUserForSomePreferences

Figura 6.17: Descripción de las plantillas durante la recuperación

dentro del subárbol definido por cada característica. Por ejemplo, imaginemos que los individuos r_a y r_b representan a los recomendadores A y B. Por cada característica, ambas serán reconocidas bajo alguno de los conceptos definidos en ese subárbol. En el caso de la característica "ByNavigationType", serían clasificados en alguno de los siguientes conceptos: "NavigationByAsking", "NavigationByProposing" u "OtherNavigation". Si fueran clasificadas bajo el mismo concepto su similitud sería 1. En el caso de que fueran clasificadas en diferentes conceptos utilizaríamos alguna de las medidas de similitud en ontologías del Capítulo 5 para obtener un valor numérico que definiera su cercanía (en este caso utilizamos de nuevo la función coseno). Este proceso se repetirá por cada una de las características globales y finalmente se calculará una media de todos los valores de similitud obtenidos.

6.4.4 Ejemplo de recuperación de plantillas

Una vez explicado el funcionamiento de la recuperación de plantillas desde el punto de vista teórico, veamos un ejemplo real donde se aplican los mecanismos propuestos.

Como se indicó en la Sección 6.4.2.1 el proceso de recuperación se inicia sin la necesidad de introducir una consulta mediante la elección de dos plantillas de la base de datos. Ambas plantillas se seleccionan siguiendo el algoritmo planteado en el sistema Expert Clerk de Shimazu (2002). Para ello se obtiene un caso ficticio que representa la media de la base de casos. A partir de él se seleccionan los casos (plantillas) que sean más similares y más distintos a este caso medio. La Figura 6.17 muestra la descripción mediante

pares atributo-valor del caso medio calculado y las dos plantillas recuperadas. La tabla contiene el valor de los atributos por cada componente de la descripción del caso: tareas, métodos y características semánticas. El primer grupo de atributos describe la descomposición en tareas de cada plantilla. Nuestras plantillas tienen cinco tareas genéricas: "One-Off Preference Elicitation", "Retrieval", "Display" y "Iterated Preference Elicitation". Por lo tanto esos son los cinco atributos que describen la estructura en tareas. Los valores de dichos atributos serán las posibles descomposiciones de estas tareas genéricas en secuencias de tareas simples. Los atributos del grupo "Methods" describen qué método se ha utilizado para resolver cada tarea simple. Por lo tanto en este grupo habrá un atributo por cada tarea simple que aparezca en todas las plantillas. Como algunas tareas simples pueden repetirse, se crea un atributo cada vez que aparezcan en la plantilla y que estará ligado a la tarea genérica a cuya descomposición pertenecen. Finalmente, las características semánticas ("Global Features") reflejan las distintas raíces de la jerarquía de conceptos mostrados en la Figura 6.16.

El caso medio utilizado para elegir los dos sistemas mostrados inicialmente es un recomendador conversacional de tipo A donde el valor de cada atributo es el más repetido entre todas las plantillas. Este caso medio no tiene atributos semánticos ya que es un recomendador ficticio y no corresponde a ningún sistema real de la base de casos. El recommendador 6 es el más parecido a este caso medio, tal y como puede observarse en los distintos valores mostrados en la Figura 6.17. Por el contrario, el recomendador 8 es el más diferente tanto a la media como al recomendador 6 de entre todos los sistemas de la base de casos.

Una vez recuperados estos casos, se muestran al usuario a través de la interfaz mostrada en la Figura 6.14. Esta herramienta permite al usuario leer descripciones de los dos recomendadores propuestos y probarlos en funcionamiento. Una vez que el usuario ha seleccionado el que más se acerca a sus preferencias, puede elegirlo como plantilla a adaptar o pedir al sistema que le muestre otros sistemas similares a ese. Es en estos pasos de la interacción con el usuario donde se utiliza el algoritmo *Bounded Greedy* detallado en la Sección 6.4.2.2.

El elemento más importante en el proceso de recuperación de plantillas es el cómputo de similitud entre las mismas. Dicho cómputo se utiliza tanto en la selección inicial mediante el algoritmo del sistema *Expert Clerk* como en las siguientes iteraciones donde se aplica el algoritmo *Bounded Greedy*. Para aclarar el funcionamiento de esta medida, expondremos a continuación cómo se calcularía la similitud entre los dos recomendadores 6 y 8 mostrados en la Figura 6.17. Lógicamente, la similitud entre ambos recomendadores será muy baja debido a que el algoritmo *Expert Clerk* recupera los casos más diversos.

Siguiendo la fórmula descrita al principio de la Sección 6.4.3:

$$sim(r_6, r_8) = w_1 \cdot ts(r_6, r_8) + w_2 \cdot mc(r_6, r_8) + w_3 \cdot gf(r_6, r_8)$$

la similitud entre tareas sería:

$$ts(r_6, r_8) = 0/4$$

ya que no existe ninguna descomposición común en tareas simples.

Para ilustrar la similitud entre métodos simplificaremos obviando la utilización de la medida coseno y estimaremos un valor de 1 si los métodos son iguales y 0 si son diferentes. Esta simplificación se realiza para mejorar la explicación, aunque el sistema real sí que utiliza la similitud coseno entre instancias de la ontología. De esta forma la fórmula de similitud de métodos devolvería:

$$ms(r_6, r_8) = 4/17$$

al haber 4 métodos en común entre las 17 tareas simples.

Por último, al calcular la similitud de acuerdo a las características globales haremos la misma simplificación obviando la función coseno y obtendríamos el valor:

$$gf(r_6, r_8) = 3/6$$

Juntando todo en la fórmula y suponiendo unos pesos de $w_1 = 0{,}1, w_2 = 0{,}4, w_3 = 0{,}5$ (cuya razón explicaremos en la próxima sección), obtendríamos una similitud final de:

$$sim(r_6, r_8) = 0{,}1 \cdot 0 + 0{,}4 \cdot 0{,}23 + 0{,}5 \cdot 0{,}5 = 0{,}34$$

Una vez detallado con un ejemplo el funcionamiento del algoritmo de recuperación de plantillas, la siguiente sección explica cómo se obtuvieron de forma experimental los pesos w_1, w_2, w_3 de la fórmula de similitud junto con otros parámetros involucrados en el proceso.

6.4.5 Evaluación y ajuste de parámetros

La evaluación del algoritmo de recuperación nos permitió obtener los valores idóneos para los pesos de la fórmula de similitud y averiguar qué estrategia de gestión de las preferencias del usuario era la más adecuada. Para ello se realizó un estudio *leave-one-in* donde uno de los sistemas de nuestra base de casos se tomaba como el objetivo del usuario. Para simular las preferencias del usuario se eliminaban algunos de los atributos del caso que se había elegido como objetivo y se tomaba esa descripción parcial como el sistema a recuperar.

Se eligieron 6 recomendadores representativos para actuar como objetivos (2 tipo recomendación única, 2 tipo conversacional A y otros dos tipo

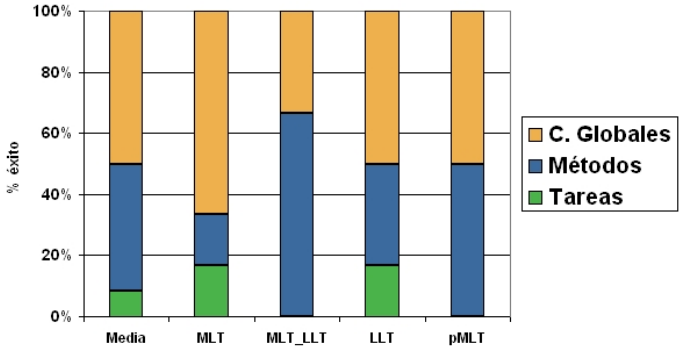


Figura 6.18: Comparación empírica de las distintas medidas de similitud entre plantillas

conversacional B). La eliminación de los atributos se hizo de forma aleatoria y cada proceso de recuperación se repitió 20 veces para permitir la elección de un conjunto de atributos distintos en cada iteración.

El experimento medía el número de pasos necesarios para recuperar el mismo recomendador gracias a la herramienta. Obviamente el número de pasos al usar el 100 % de los atributos era siempre 0, pero se incrementaba al utilizar tan sólo el 75 %, 50 % y 25 % de los mismos.

En una primera etapa se utilizaron las tres medidas de similitud por separado: tareas, métodos y características globales. Además, se probaron las diferentes estrategias de gestión de las preferencias de usuario (Sección 6.4.2.3): MLT, LLT, pMLT y MLT-LLT. Los resultados se muestran en la Figura 6.18 donde se refleja el porcentaje de consultas que resultaron en un menor número de pasos para cada una de las tres medidas y las 4 estrategias de gestión de preferencias. Como puede observarse, la similitud mediante características globales fue la medida que alcanzó mejores resultados, seguida de cerca por la similitud entre métodos. Sin embargo, la similitud basada en la estructura de tareas resultó ser poco eficiente. Haciendo una media entre los valores obtenidos podríamos afirmar que la similitud entre tareas obtuvo mejores resultados un 10 % de las veces, frente a un 40 % de la similitud por métodos y un 50 % de la similitud por características globales. Como podría esperarse, esta última medida obtiene mejores resultados al tratarse de un mecanismo de similitud rico en conocimiento.

La segunda etapa del experimento consistió en comprobar nuestra hipótesis de que la mejor medida de similitud sería una media ponderada de las tres medidas implementadas de acuerdo a los pesos obtenidos en la primera etapa. Esta hipótesis se confirmó en los experimentos mostrados en la Figura 6.19 (derecha) donde esa combinación de pesos obtiene mejores resultados que varias otras (70 %-15 %-15 %, 15 %-70 %-15 %, 15 %-

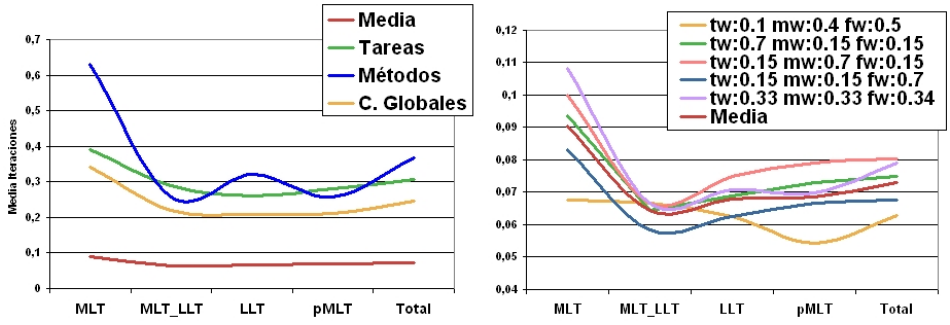


Figura 6.19: Combinaciones óptimas de los pesos para las medidas de similitud entre plantillas

15 %-70 % y 33 %,33 %,34 %). Además, en la parte izquierda de dicha figura se comparan los resultados obtenidos al utilizar la combinación de pesos $w_1 = 0{,}1, w_2 = 0{,}4, w_3 = 0{,}5$ con respecto a las tres aproximaciones por separado. Por lo tanto, la mejor configuración de nuestro mecanismo de recuperación sería el de los pesos $w_1 = 0{,}1, w_2 = 0{,}4, w_3 = 0{,}5$ junto con la estrategia pMLT ya que es ahí donde se obtiene el mínimo en la gráfica de la parte derecha.

6.5 Adaptación de plantillas

El proceso de composición de sistemas CBR empieza con la recuperación de la plantilla más similar tal y como se ha expuesto en la sección anterior. A continuación nuestra herramienta de composición permite adaptar la plantilla recuperada a las necesidades del diseñador. Las plantillas recuperadas por nuestro sistema de "Recuperación probando" son sistemas finales donde las tareas tienen métodos asignados. Sin embargo, la herramienta de composición permite tanto adaptar un sistema ya implementado cambiando los métodos que resuelven las tareas, o diseñar el sistema desde cero asignando métodos a plantillas finales que no estuvieran instanciadas en sistemas.

Antes de pasar a la etapa de adaptación, la herramienta debe obtener los métodos disponibles para resolver las tareas. Este proceso conlleva el análisis de las anotaciones semánticas en el código de jCOLIBRI 2 y la generación de la representación mediante CBROnto que se expuso en la Sección 6.3.1. Una vez realizado esta generación de la representación semántica de los métodos nuestro sistema estará listo para adaptar las plantillas y comprobar su coherencia y validez. Estas comprobaciones se basarán en dicha representación semántica y en los mecanismos de razonamiento de las lógicas descriptivas. Por ello, la herramienta de composición también utiliza nuestra librería OntoBridge para gestionar y razonar sobre la ontología CBROnto.

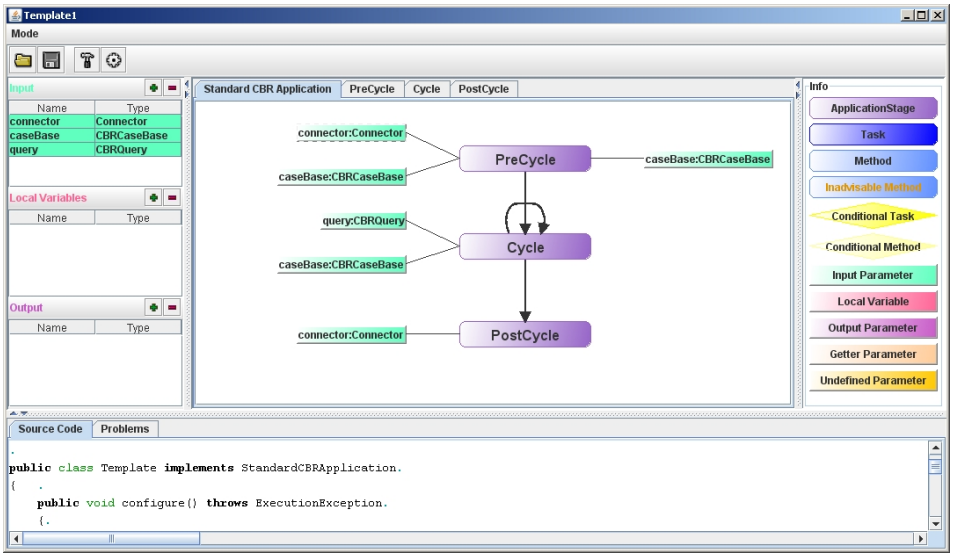


Figura 6.20: Vista genérica de la herramienta de adaptación de plantillas

La Figura 6.20 muestra una captura de la herramienta donde podemos distinguir los distintos paneles que la componen. Como se explicó durante la discusión sobre la representación de los métodos, debemos distinguir entre los parámetros de entrada y salida de cada método visto como entidad única, y las entradas y salidas de la composición a nivel global. Esas variables globales a nivel de composición son las mostradas en la parte izquierda de la herramienta: entradas (input), salidas (output) y variables globales (local variables). Estos paneles permiten crear dichas variables y asignarles un tipo concreto. Las variables definidas en los paneles conformarán el contexto de la aplicación y se utilizarán para configurar el flujo de datos entre los distintos métodos de la plantilla. Las variables locales servirán para enlazar la salida de un método con la entrada de otro. Sin embargo, las variables de entrada deberán ser definidas por el usuario y pueden considerarse como parámetros de configuración del sistema generado.

En la parte central se encuentran las diferentes partes de la plantilla: preciclo, ciclo, postciclo y una vista global de la aplicación. Las tres primeras partes serán editables por el diseñador, mientras que la vista global (mostrada en la Figura 6.20) refleja la estructura global de la aplicación CBR a generar y la interconexión de las tres partes que la componen (preciclo, ciclo y postciclo). Esta vista global de la plantilla no es editable ya que refleja el comportamiento definido por la interfaz `StandardCBRApplication`. Recordando la Sección 3.2.1.2 y más concretamente el Listado 3.5 (pág. 98), esta interfaz establecía la división de las aplicaciones CBR en jCOLIBRI. Cualquier aplicación generada con el armazón debería implementar esta interfaz

o alguna subclase suya. Por lo tanto, la vista genérica de la herramienta muestra el comportamiento de esta inferfaz y define sus entradas, salidas y variables globales a nivel de aplicación CBR completa. Como ya se planteó en dicha sección, una aplicación CBR en jCOLIBRI recibirá un objeto *base de casos* que se rellenará con los casos obtenidos desde el medio de persistencia. La obtención de dichos casos se realiza mediante un *conector*. Finalmente el sistema recibe la *consulta* del usuario. Estos tres elementos son los mostrados como entradas en el panel izquierdo de la captura.

Una vez ejecutado el preciclo la base de casos contendrá los distintos casos del sistema. Este mismo objeto es el utilizado por el ciclo de la aplicación CBR. Este hecho se refleja en el esquema de la vista general ya que el parámetro `caseBase` está asignado a la entrada y salida del preciclo. A continuación, este mismo parámetro aparece en la entrada del ciclo, significando que la salida del preciclo estará conectada con la entrada del ciclo.

Finalmente, el postciclo se encargará únicamente de cerrar el conector. Por ese motivo lo recibe como parámetro.

Como es difícil definir la salida de un sistema CBR debido a los distintos tipos de aplicaciones y usos de las mismas, se ha decidido no representarlas explícitamente. Es por ello que el panel izquierdo aparece vacío. Deberá ser el diseñador quien defina estas salidas directamente dependiendo del tipo de aplicación.

Tal y como se puede observar en la Figura 6.20, nuestro editor representa los parámetros de entrada a la izquierda del componente o método, y las salidas a la derecha. Cuando se asigna la misma variable como parámetro de salida y entrada de dos componentes distintos, estos quedan conectados por ella, significando que el valor producido en la salida se asignará a la entrada del siguiente.

En el caso de la Figura 6.20 los elementos interconectados son etapas de la aplicación CBR. Cuando el usuario edite por separado dichas partes (preciclo, ciclo y postciclo), las entradas y salidas estarán ligados a métodos y no a etapas de la aplicación.

La parte derecha de la herramienta muestra un panel de información contextual. Por defecto muestra la información que aparece en la Figura 6.20 pero al pasar el ratón sobre los distintos elementos se visualiza información concreta sobre dicho elemento. La información por defecto muestra una leyenda sobre los elementos que aparecen en la plantilla. Los distintos tipos de parámetros aparecerán siempre con un color determinado dependiendo del tipo de variable asignada. Ese mismo color es el utilizado en el panel de definición de variables que aparece en la parte izquierda de la ventana.

La información contextual mostrada al pasar el ratón sobre cada elemento se obtiene de las anotaciones semánticas de los métodos que se han almacenado en CBROnto. De esta forma es posible acceder a las distintas

descripciones añadidas por el desarrollador sobre los métodos y parámetros, a la información sobre los tipos de los parámetros (obligatorios o recomendables), etcétera.

Finalmente, la herramienta muestra en la parte inferior dos paneles. El primero contendrá el código fuente generado una vez realizada una composición de métodos libre de errores. Y el segundo panel muestra los distintos problemas de incompatibilidad en la composición. Cada vez que el usuario realice una acción de edición, el sistema utilizará la información semántica sobre los métodos para comprobar la validez de la plantilla. Los problemas encontrados aparecerán en este panel junto con su tipo, origen y descripción.

Una vez descrita la herramienta a nivel genérico podemos centrarnos en la parte más importante de la misma: la composición de los métodos que resuelven las distintas tareas de las plantillas y la comprobación semántica de dichas composiciones. La exposición de ambos procesos se llevará a cabo en las próximas secciones 6.5.1 y 6.5.2. Finalmente se explicará el proceso de generación de la aplicación final a partir de la composición realizada en la sección 6.5.3.

6.5.1 Composición de métodos

La adaptación de las plantillas se realiza asignando o substituyendo los métodos que resuelven cada tarea del preciclo, ciclo o postciclo. El editor utilizará la información contenida en CBROnto tanto para sugerir como para detectar errores en la asignación de los métodos y sus parámetros.

En el preciclo únicamente encontramos una tarea denominada "LoadCaseBase" que se resuelve con el método "LoadCaseBaseMethod" encargado de cargar los casos desde el medio de persistencia. Este método recibe y devuelve los mismos parámetros que el preciclo mostrado la Figura 6.20. Por su parte, el postciclo también se compondrá normalmente de una única tarea "CloseConnector" que se resuelve con un método que recibe el conector para cerrarlo y liberar los recursos. Estos dos métodos encargados de cargar la base de casos y cerrar el conector son métodos creados específicamente para la herramienta de composición y en su interior contienen el código que se utiliza comúnmente para realizar estas dos tareas y que se incluye explícitamente al utilizar el armazón directamente (programando sobre el API en vez de usar la herramienta de composición).

La siguiente etapa es la más importante de la plantilla: el ciclo. La Figura 6.21 muestra el ciclo de un sistema de recuperación única (single-shot) que utilizaremos como ejemplo para explicar las características del editor. En este caso, la plantilla está incompleta para poder ilustrar los mecanismos de comprobación semántica incorporados y que se detallarán en la próxima sección. Por ello, algunos métodos y parámetros aparecerán con color de fuente rojo. Por ahora obviaremos este hecho en la explicación.

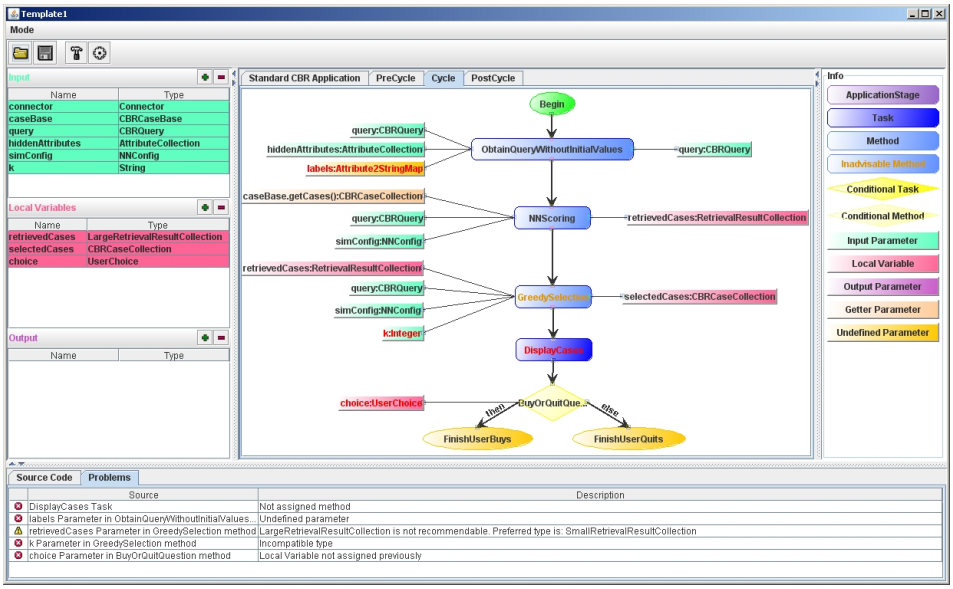


Figura 6.21: Proceso de adaptación de plantillas

En el panel de variables de entrada encontramos las tres variables que se mostraban en la vista global del sistema: conector (`connector`), base de casos (`caseBase`) y consulta (`query`). Si seguimos el flujo de la plantilla (empezando por la tarea *begin*) encontramos que a la primera tarea "Obtain cases" se le ha asignado el método `ObtainCasesWithoutInitialValues`, la siguiente tarea es "Scoring" y tiene asignado el método `NNScoring`. A continuación estaría la tarea "Selection" resuelta mediante el método `GreedySelection`. Le sigue la tarea "Display" a la que no se ha asignado ningún método. Finalmente, la tarea "BuyOrQuit" ha sido resuelta mediante el método `BuyOrQuitMethod`.

La asignación de métodos se realiza pulsando sobre cualquiera de las tareas. De este modo aparecerá el cuadro de diálogo mostrado en la Figura 6.22. En este caso estamos resolviendo la tarea "Scoring" y el sistema busca en la ontología aquellos métodos que la resuelven. Esta información se habrá obtenido de las anotaciones del código fuente y se representará en CBROnto mediante la propiedad *solves* que enlaza cada métodos con las tarea que resuelve. El cuadro de diálogo para la elección de métodos muestra la descripción del método (también obtenida de las anotaciones) y permite consultar su representación representación OWL. Esta representación es exactamente la que se presentaba como ejemplo en la Figura 6.13.

Una vez elegido el método que resolverá una tarea, la herramienta añade los parámetros correspondientes en el grafo de la plantilla. Inicialmente, dichos parámetros tendrán asignados variables con el nombre por defecto que se indicara en su anotación semántica. Si dichas variables ya estuvieran

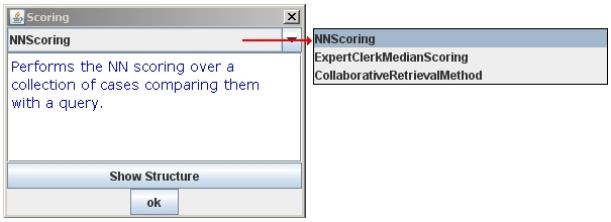


Figura 6.22: Elección del método para resolver una tarea

definidas en el panel de definición de variables que aparece en la parte izquierda de la ventana, se mostrará su tipo (entrada, salida o variable local) mediante el color correspondiente. Si no se encontraran definidas, el editor las mostrará como parámetros indefinidos (*Undefined parameter*). En el ejemplo, podemos observar este hecho en el parámetro `labels` del método `ObtainQueryWithoutInitialValues`.

El usuario podrá definir las variables que desee a través de los botones de los tres paneles. Al pulsar en el botón de inserción (indicado con un signo +) se introducirá el nombre de la variable y a continuación se elegirá su tipo dentro de la jerarquía de tipos definida en CBROnto según se muestra en la Figura 6.23. Esta jerarquía es la que se utilizó para indicar los tipos de los métodos en las anotaciones del código fuente.

Para asignar una variable a un parámetro simplemente se pulsa sobre éste y el sistema mostrará un menú despegable con las variables definidas actualmente que sean compatibles. Esta compatibilidad se basa en la comparación de los tipos de datos de los parámetros y variables dentro de la jerarquía conceptual de CBROnto. Una variable podrá asignarse a un parámetro si su tipo es igual o un subconcepto del tipo del parámetro. Además, este mecanismo tiene que extenderse para poder utilizar los métodos accesores que permiten acceder a miembros de una variable. Por ejemplo, la Figura 6.24 (pág. 245) muestra los valores compatibles encontrados para ser asignados al parámetro `cases` de tipo `CBRCaseCollection` en el método `NNScoring`. Aquí, la variable `selectedCases` estaría definida como variable local en el contexto. Además, el contexto incluye la variable `caseBase` de tipo `CBRCaseBase` a la que se añadió una anotación (mostrada en el listado 6.4) para indicar que tiene un método accesor `getCases()` que devuelve la lista de casos (también de tipo `CBRCaseCollection`). Gracias a esta información sobre el método accesor que se ha incluido en CBROnto, el sistema puede inferir que `caseBase.getCases()` devuelve un valor compatible con el parámetro y sugerírselo al diseñador para su asignación.

Si no existe ninguna variable compatible con el parámetro, el diseñador deberá crearla bien directamente mediante los paneles de variables o a través del menú contextual que aparece al pulsar con el botón derecho sobre cual-

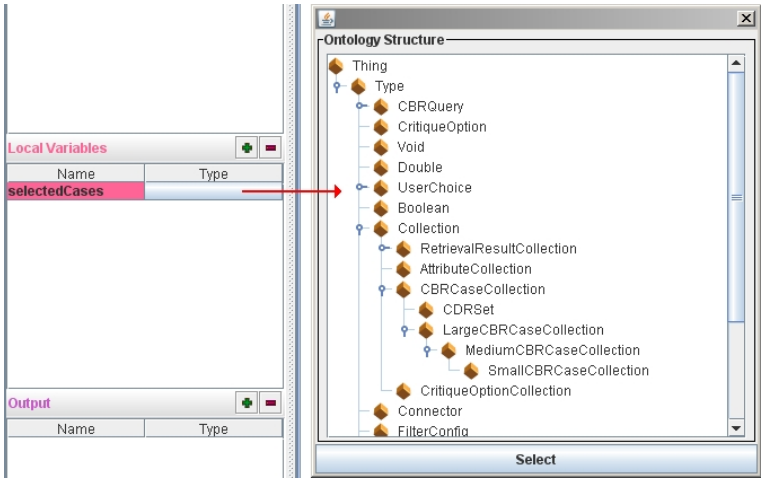


Figura 6.23: Jerarquía de tipos en CBROnto

quier parámetro. Este menú se muestra en la Figura 6.25 y sirve de acceso rápido para crear las variables. De esta forma, se preguntará por el nombre de la variable y se mostrará la jerarquía de tipos de la Figura 6.23.

La asignación de tipos a las variables puede ser utilizado por el diseñador para especificar restricciones. Por ejemplo, volviendo a la Figura 6.21, el método `NNScoring` devuelve un parámetro de tipo `RetrievalResultCollection` al que se le ha asignado la variable `retrievedCases` que ha sido tipada por el diseñador como `LargeRetrievalResultCollection`. Mediante esta especificación del tipo de la variable el diseñador está añadiendo una restricción a la composición. En el ejemplo esta restricción implica que el siguiente método `GreedySelection` no sea recomendable –por eso aparece en color naranja– ya que en su parámetro de entrada se ha establecido que prefiere una colección de casos recuperados pequeña (`SmallRetrievalResultCollection`).

La Figura 6.21 incluye diversos elementos –tanto métodos como variables– cuyo color del texto aparece en rojo o naranja. Esta variación se utiliza en la herramienta para indicar errores o avisos sobre la composición actual del sistema. A continuación se detallará cómo se realizan dichas comprobaciones gracias a las anotaciones semánticas de los métodos.

6.5.2 Comprobación semántica de la composición de métodos

La representación de los métodos en CBROnto permite realizar varias comprobaciones sobre la validez de los sistemas diseñados mediante la herramienta. Estas comprobaciones pueden ser errores o recomendaciones. Ambos tipos

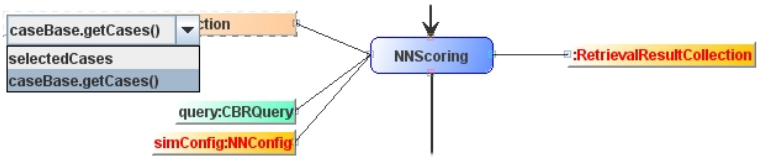


Figura 6.24: Ayuda al usuario en la asignación de variables a los parámetros

de comprobaciones aparecerán mostrados en el panel inferior de la ventana identificado como "Problems". Este panel indicará el tipo de problema, el elemento que lo origina y una descripción sobre el mismo.

A continuación se indican las comprobaciones realizadas por el sistema, quedando todas ellas ejemplificadas en la Figura 6.21:

Tareas sin método asignado. Esta comprobación no utiliza realmente la información en CBROnto ya que únicamente comprueba que cada tarea tenga asignada un método. En la figura, esta comprobación queda reflejada en la tarea "DisplayCases".

Parámetros sin variable asignada. Aquí se comprueba que cada parámetro tenga asignado una variable del contexto. Por defecto cada parámetro tiene asignada una variable con su mismo nombre (que se definió durante la anotación del método). Si dicho identificador no estuviera definido como variable, el sistema indicará dicho error. En la Figura 6.21 esto se refleja en el parámetro `labels` del método `NNScoring`.

Parámetros con variable asignada incompatible. La jerarquía de tipos de CBROnto permite comprobar la validez de las variables asignadas a cada parámetro. Una variable sólo será válida si su tipo es equivalente o subconcepto del tipo del parámetro dentro de CBROnto. En este caso estaríamos refiriéndonos al tipo obligatorio (restricción fuerte) del parámetro. Para realizar esta comprobación se utilizará el razonador de Lógicas Descriptivas. Como ejemplo de esta comprobación, se ha asignado intencionadamente el tipo `String` a la variable `k` utilizada en el método de selección Greedy. El sistema muestra un error ya que dicho parámetro debe ser de tipo `Integer` que es incompatible con `String`.

Variable Local no asignada previamente. Las variables locales son internas a cada composición y permiten enlazar un parámetro de salida con otro de entrada. Por lo tanto, si se ha asignado una variable local como entrada de un método y ésta misma variable no está asignada en

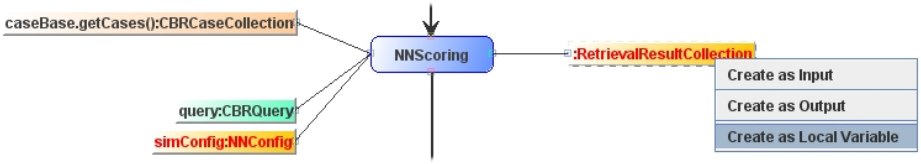


Figura 6.25: Ayuda al usuario en la creación de variables

un parámetro de salida, el sistema generará un error avisando al usuario. La figura incluye un ejemplo de este error en el parámetro `choice` del método `BuyOrQuitMethod`.

Método no recomendable. Al explicar las restricciones de tipo en las entradas de los métodos, se diferenció entre las restricciones fuertes (obligatorias) y las restricciones débiles (recomendables). Las restricciones débiles reflejaban recomendaciones del desarrollador sobre el tipo de parámetro a recibir y solían estar relacionadas con el rendimiento del método. Como ejemplo de estas restricciones se detallaba el método `GreedySelection` que no funciona apropiadamente con colecciones de casos grandes. Si se asignara como entrada una colección de casos grande, el método seguiría funcionando pero no de forma óptima. Este hecho queda reflejado también como una de las comprobaciones que hace la herramienta sobre los sistemas. Pero en este caso no se indica como un error sino como un aviso, y por ello aparece mostrado con el texto en naranja. Para realizar esta comprobación se utiliza el razonador de Lógicas Descriptivas y la jerarquía de tipos de CBROnto, pero en este caso se utiliza el tipo recomendado por la anotación del parámetro. En la figura, el error se debe a que el tipo de la variable `retrievedCases` es `LargeRetrievalResultCollection`. Como la entrada recomendada por el método es de tipo `SmallRetrievalResultCollection`, el sistema infiere que no son compatibles.

Una vez realizada la composición de métodos, la herramienta dispone de toda la información necesaria para generar el código del sistema diseñado e incluso ejecutarlo dinámicamente. Estas opciones son las que se detallan en la siguiente sección.

6.5.3 Generación de la aplicación

Gracias a la información almacenada en la representación OWL de la plantilla es posible generar el código de la aplicación diseñada. De esta forma se ofrece la misma funcionalidad típica de los *generadores de aplicaciones* (ver Sección 2.4.2) que también se incluía en jCOLIBRI 1.

Para generar el código de la aplicación, se necesitan dos elementos:

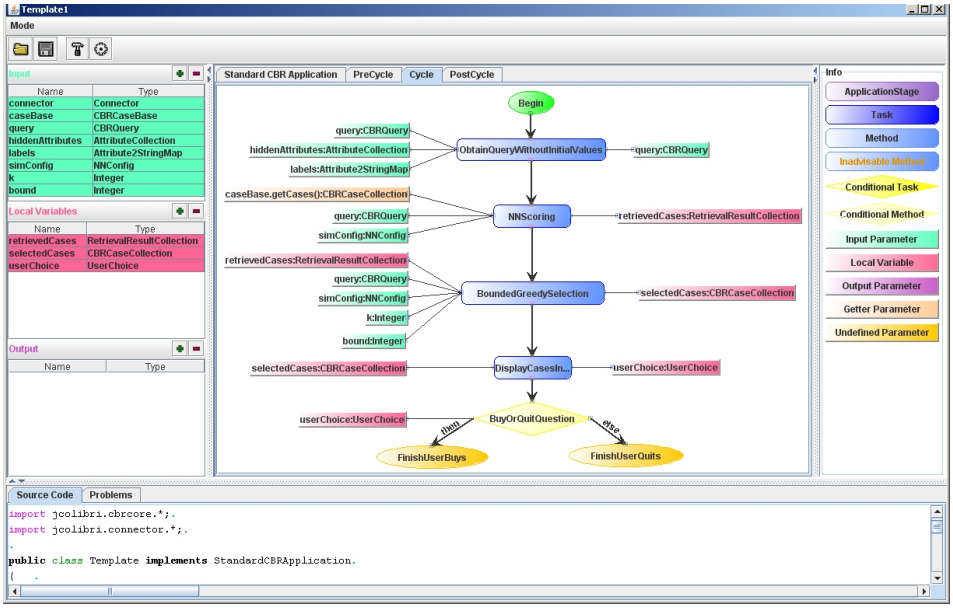


Figura 6.26: Diseño correcto de un sistema Single-Shot

- El *flujo de ejecución* que está almacenado en la plantilla (secuencias, bucles, condicionales, etc.), y
- El *flujo de datos* entre los diferentes parámetros de una plantilla implementada que se configura en el editor de sistemas.

Mediante esta información, nuestra herramienta puede generar una aplicación semicompleta donde sólo falta configurar los valores de las variables de entrada del sistema. Para ilustrar esta funcionalidad supongamos que hemos completado el sistema de la Figura 6.21 creando las variables que faltaban, sustituyendo el método `GreedySelection` por `BoundedGreedySelection` ya que este último puede trabajar con colecciones de casos grandes, y además hemos resuelto la tarea `Display` con el método `DisplayCasesInTableBasic`. La versión completa y correcta del sistema aparece mostrada en la Figura 6.26.

La generación de código creará una clase que implemente la interfaz `StandardCBRApplication` ya que esta interfaz era la que definía la estructura global de la plantilla. El código generado aparecerá en el panel inferior del editor, donde el usuario podrá completarlo para definir los valores de los parámetros de entrada. Esta configuración se llevará a cabo en el método `configure()`.

El siguiente listado muestra el código semicompleto (sin la asignación de variables de entrada) que generaría la herramienta para el sistema de la

Figura 6.26:

Listado 6.5: Aplicación semicompleta generada por el editor

```
package jcolibri2.templates.generated;

public class System1 implements StandardCBRApplication {

    // Input variables
    private Connector connector;
    private CBRCaseBase caseBase;
    private CBRQuery query;
    private Collection<Attribute> hiddenAttributes;
    private Map<Attribute, String> labels;
    private NNConfig simConfig;
    private Integer k;
    private Integer bound;

    // Local variables
    private Collection<RetrievalResult> retrievedCases;
    private Collection<CBRCase> selectedCases;
    private UserChoice userChoice;


    public void configure() throws ExecutionException {

        // Input variables
        connector = //COMPLETE HERE
        caseBase = //COMPLETE HERE
        hiddenAttributes = //COMPLETE HERE
        labels = //COMPLETE HERE
        simConfig = //COMPLETE HERE
        k = //COMPLETE HERE
        bound = //COMPLETE HERE

        // Local variables are initialized by default.
        // COMPLETE HERE if necessary
        retrievedCases = new ArrayList<RetrievalResult<();
        selectedCases = new ArrayList<CBRCase>();
        userChoice = new UserChoice();

    }

    public void cycle(CBRQuery query) throws ExecutionException {
        this.query = query;
        sequence1();
    }

    private void sequence1() {
        ObtainQueryWithFormMethod.obtainQueryWithoutInitialValues(
```

```
            query, hiddenAttributes, labels);
        retrievedCases = NNScoringMethod.evaluateSimilarity(caseBase.
            getCases(), query, simConfig);
        selectedCases = BoundedGreedySelection.boundedGreddySelection(
            retrievedCases, query, simConfig, k, bound);
        userChoice = BuyOrQuit.displayCasesInTableBasic(selectedCases)
            ;
        if(ConditionalMethods.buyOrQuit(userChoice))
            // FinishUserBuys
        else
            // FinishUserQuits
    }

    public void postCycle() throws ExecutionException {
        WrapperMethods.closeConnector(connector);
    }

    public CBRCaseBase preCycle() throws ExecutionException {
        return WrapperMethods.loadCasesMethod(connector, caseBase);
    }

    public static void main(String[] args) {
        StandardCBRApplication recommender = new System1();
        try
        {
            recommender.configure();
            recommender.preCycle();
            CBRQuery query = //COMPLETE HERE
            recommender.cycle(query);
            recommender.postCycle();
        } catch (Exception e)
        {
            org.apache.commons.logging.LogFactory.getLog(System1.class
                ).error(e);

        }
    }
}
```

Como puede observarse en el listado anterior, el diseñador únicamente tiene que inicializar las variables de entrada sustituyendo el comentario `COMPLETE HERE` con el código apropiado.

Una vez que el código haya sido completado, la herramienta ofrece la posibilidad de compilar y ejecutar el código directamente para comprobar el correcto funcionamiento del sistema recomendado.

6.6 Conclusiones

En este capítulo se ha presentado la herramienta de composición de sistemas CBR de jCOLIBRI 2. Esta herramienta se enmarca dentro de la arquitectura del armazón en la capa dirigida a usuarios diseñadores (mostrada en la Figura 3.11 de la página 94). El mecanismo de diseño propuesto se basa en la reutilización de plantillas que abstraen el comportamiento de los sistemas CBR. La reutilización de las plantillas se basa en el mismo principio que el Razonamiento Basado en Casos: problemas similares tienen soluciones similares. Extrapolando esta idea al diseño de sistemas podemos suponer que los sistemas CBR tenderán a recurrir y tendrán implementaciones similares.

Como prueba de concepto, la herramienta se ha limitado inicialmente a sistemas CBR de recomendación. Para ello se han definido varias plantillas que abstraen el comportamiento de una amplia familia de recomendadores. A partir de ellas se han generado diversos sistemas reales que las implementan. Las plantillas representan el flujo de ejecución de la aplicación mediante tareas siguiendo las ideas de la metodología KADS que también se utilizó en jCOLIBRI 1. La implementación de dichas plantillas en sistemas reales se realizará asignando un método que resuelva cada tarea.

Tanto las plantillas como los sistemas que las implementan conforman la base de casos utilizada por la herramienta de composición. Las primeras secciones del capítulo muestran la organización jerárquica de esta base de casos y detallan el comportamiento de las plantillas de sistemas de recomendación junto con los distintos métodos incluidos en el armazón que permiten su instanciación en sistemas reales.

La representación de las plantillas y los métodos se basa en los estándares de los Servicios Web Semánticos OWL-S y WSMO. Dichos estándares utilizan ontologías para definir y almacenar el comportamiento de los servicios, lo que nos permite aprovechar la ontología CBROnto que aparecía en el sistema COLIBRI original. Esta ontología ha sido rediseñada para permitir la representación del flujo de ejecución de los sistemas, su recuperación y la adaptación posterior de los métodos.

jCOLIBRI 2 incluye una herramienta que permite diseñar gráficamente las plantillas y almacenarlas mediante CBROnto. Estas plantillas serán luego instanciadas por otra herramienta que permite la asignación de métodos a las tareas.

El proceso de reutilización de plantillas se divide en dos etapas: recuperación y adaptación. Para la recuperación se ha implementado un recomendador que permite al diseñador elegir el sistema más cercano a sus requisitos. El recomendador interactúa con el diseñador ofreciéndole ejecutar los sistemas para comprobar su funcionamiento real. Este novedoso mecanismo de recupe-

ración ha sido bautizado como "Recuperar probando" (Retrieval-by-Trying) y evita la utilización de algún lenguaje descriptivo que sería demasiado complejo y poco intuitivo. La elección de los sistemas presentados al usuario se basa en una medida de similitud compuesta por tres elementos distintos. La similitud sobre la estructura de tares indica qué hace el sistema y la similitud de los métodos utilizados mide cómo se implementa la funcionalidad del sistema. Por último, se utiliza la similitud entre una serie de características semánticas establecidas por el diseñador que mezclan aspectos sobre la estructura y funcionalidad de la plantilla. Estas características semánticas se definen en CBROnto y permiten calcular la similitud entre sistemas gracias a los mecanismos de reconocimiento de instancias de los razonadores de Lógicas Descriptivas.

Una vez recuperado el sistema más similar a los requisitos del diseñador se pasa a la fase de adaptación. En este caso, la herramienta sugiere los métodos más apropiados para resolver cada una de las tareas y comprueba la coherencia semántica de la composición realizada. Para realizar estas comprobaciones se requiere una representación semántica de los métodos del armazón que también se define mediante CBROnto. Aquí aparece el punto de unión entre las dos capas de jCOLIBRI 2: obtener una representación semántica desde el código fuente utilizado por los desarrolladores. Esta cuestión se resuelve gracias a las anotaciones Java. Las anotaciones permiten incorporar meta-información al código que puede ser procesada en tiempo de ejecución. Este mecanismo nos posibilita definir una serie de anotaciones que se añaden a los métodos del armazón y que son procesadas por la herramienta de adaptación de plantillas para generar la representación semántica de dichos métodos como instancias de CBROnto.

Esta representación de los métodos incluye restricciones de ejecución (pre-/postcondiciones) en base a sus parámetros de entrada y salida. El tipo de los parámetros estará almacenado como una jerarquía conceptual en CBROnto, lo que permitirá razonar sobre su compatibilidad. Además, se incluirán restricciones débiles que serán tomadas como recomendaciones sobre la composición de los parámetros y que servirán para sugerir al usuario los métodos más apropiados para resolver una tarea.

El flujo de datos entre los parámetros de los métodos se establece mediante una serie de variables que forman el *contexto* de la composición. El contexto ya aparecía en las versiones anteriores del armazón y suponía un espacio intermedio para el encadenamiento de los métodos. En la nueva herramienta de composición, las variables son de tres tipos: variables de entrada que permiten configurar la composición realizada, variables intermedias utilizadas únicamente para encadenar métodos y variables de salida que representarían los valores devueltos por cada composición.

A grandes rasgos, la funcionalidad ofrecida por la herramienta permite:

sugerir métodos para la resolución de tareas, comprobar la compatibilidad de las variables asignadas a los parámetros e inferir la idoneidad de los métodos. Todos estos procesos se implementan gracias al razonamiento sobre las representaciones semánticas en CBROnto.

La herramienta de adaptación también permite generar el código de la aplicación diseñada una vez que se ha realizado una composición correcta. Para ello se utiliza la información sobre el flujo de ejecución almacenada en la plantilla y la información sobre el flujo de datos establecido durante la asignación de métodos.

El mecanismo de composición aquí presentado podría denominarse *diseño basado en plantillas*, en contraposición al *diseño basado en métodos* utilizado al componer los métodos manualmente cuando se programa directamente sobre el código fuente. El siguiente capítulo presenta una evaluación del armazón para comprobar las ventajas, posibilidades y aceptación de ambas opciones.

Capítulo 7

Evaluación de jCOLIBRI

– Después de tanto trabajo, ¿funcionará mi martillo? Ya sé, se lo prestaré a mis compañeros a ver qué les parece.

En este breve capítulo se pretende analizar la aceptación de jCOLIBRI por parte de los usuarios. Para ello se han utilizado dos métodos distintos. El primero consiste en analizar las estadísticas de las descargas del armazón y las diferentes visitas a la página web para medir su repercusión a nivel internacional. También se incluye una recopilación de las diferentes universidades, proyectos o empresas donde se utiliza a nivel académico o de investigación.

El segundo método consiste en una evaluación directa mediante una serie de encuestas realizadas a un amplio número de alumnos que utilizó jCOLIBRI para la realización de una práctica de un curso en la propia Universidad Complutense. Mediante estas encuestas se analiza la opinión de estos usuarios respecto a la calidad, documentación y facilidad de desarrollo con el armazón.

7.1 Repercusión internacional de jCOLIBRI

La aceptación e interés de la comunidad CBR por nuestro armazón es difícil de medir ya que es imposible realizar una evaluación directa de la opinión general. Sin embargo, hechos concretos como el número de descargas, usuarios y la presencia de jCOLIBRI en las conferencias Europea e Internacional de CBR demuestran un considerable interés por el mismo. Para intentar medir la repercusión de jCOLIBRI 2 utilizaremos dos medidas objetivas: el número de descargas, y el número y distribución de las visitas al sitio web del proyecto.

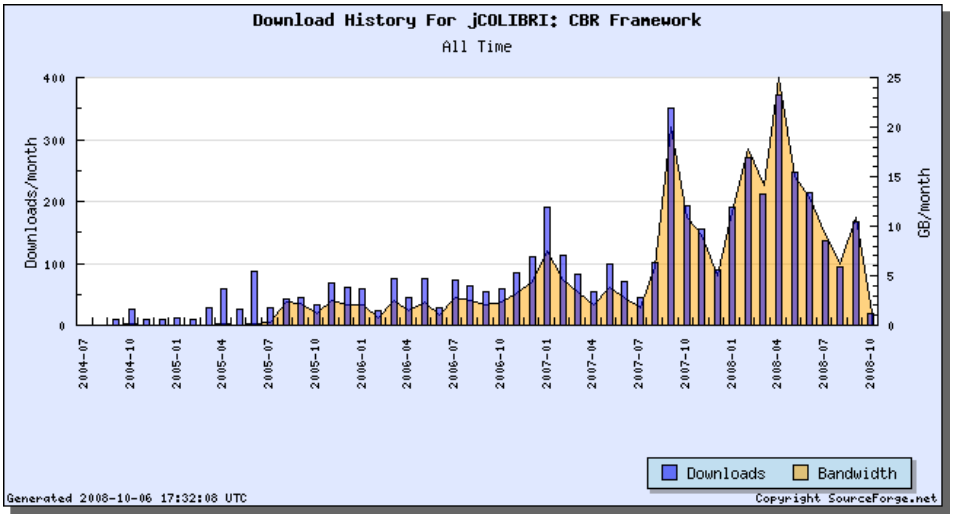


Figura 7.1: Gráfica de descargas de jCOLIBRI

Empezando por el número de descargas, la Figura 7.1 muestra una gráfica con el número de descargas del armazón[1].

jCOLIBRI acumula un total de casi 5.000 descargas[2]. Para analizar este dato respecto a cada una de las distribuciones (versiones) del armazón, hay que tener en cuenta que desde la creación del proyecto en julio de 2004 se fueron publicando diferentes versiones de prueba hasta febrero de 2005 cuando se hizo pública la primera versión estable de jCOLIBRI 1. Es en julio de 2006 cuando se publica la versión definitiva de la primera versión del armazón, mientras que jCOLIBRI 2 se publicaría en agosto de 2007.

El número de descargas acumuladas de las distintas versiones de jCOLIBRI 1 es aproximadamente 2.000. Por su parte, jCOLIBRI 2 acumula unas 3.000 descargas en poco más de un año de publicación. Este último dato supone una media de más de 200 descargas al mes para la última versión del armazón.

Además de los datos sobre las descargas directas del armazón, también se debe tener en cuenta la repercusión del la librería *OntoBridge* que fue creada como un subproyecto de jCOLIBRI para facilitar la gestión de ontologías (Capítulo 5). Esta librería ha sido publicada en proyecto independiente[3] por

[1]Esta gráfica ha sido obtenida directamente del sitio web SOURCEFORGE.NET que alberga el proyecto (http://sourceforge.net/projects/jcolibri-cbr/). Las estadísticas fueron recopiladas a 06 de octubre de 2008 por lo que las descargas de dicho mes son sólo datos parciales. Este mismo inconveniente lo encontraremos en la Figura 7.2

[2]A fecha 06/10/2008 el dato exacto son 4.773 descargas, por lo que se espera alcanzar la cifra de 5.000 en el mes de noviembre de 2008

[3] http://sourceforge.net/projects/ontobridge/

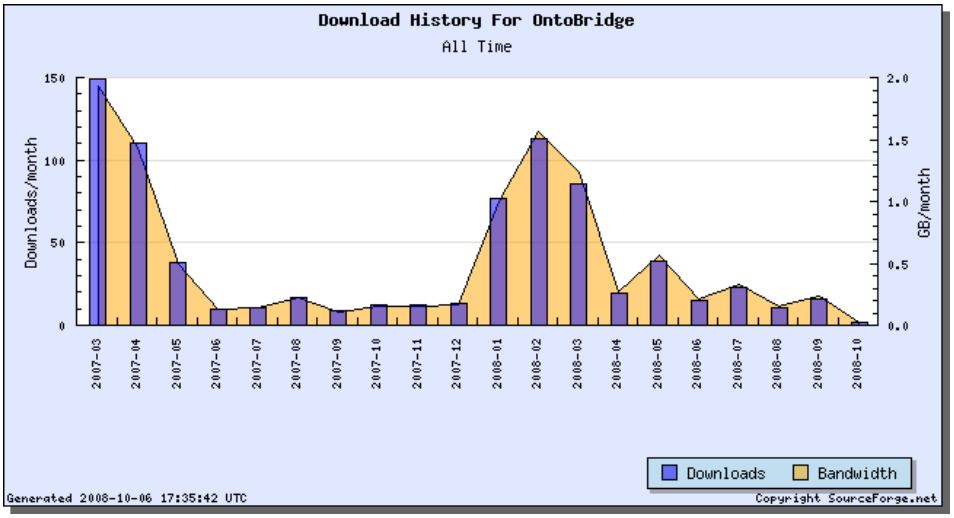


Figura 7.2: Gráfica de descargas de OntoBridge

lo que también podemos obtener su número de descargas.

OntoBridge se publicó en mayo de 2007 y desde entonces acumula unas 800[4] descargas que se muestran en la gráfica de la Figura 7.2. Esta librería se ha utilizado desde su creación en una asignatura de la Facultad de Informática de la Universidad Complutense de Madrid para la realización de prácticas en las que se necesitaba gestionar ontologías. Por lo tanto, los picos en la gráfica anterior corresponden con los periodos de realización de dichas prácticas.

Otro tipo de información que nos ayuda a medir la repercusión del armazón en la comunidad CBR es el mapa de visitas y usuarios que accedieron al sitio web de jCOLIBRI[5]. En la Figura 7.3 se muestran los gráficos de usuarios (parte superior) y visitas (parte inferior) desde febrero a julio de 2008. Como puede observarse, casi 1.400 usuarios conectaron con el sitio web realizando un total de 2.350 visitas.

La distribución de las visitas también nos aporta información sobre el uso de jCOLIBRI en la comunidad CBR. Al estudiar los orígenes de las visitas en detalle podemos comprobar que la mayoría de accesos provienen de ciudades donde existen grupos de investigación en CBR. La Figura 7.4 muestra el mapa internacional de distribución de visitas a la página web de jCOLIBRI donde aparecen 71 países distintos (también desde febrero a julio de 2008). Además, la Figura 7.5 detalla el porcentaje de accesos respecto a los 10 países

[4]Exactamente 782 descargas a 10/10/2008

[5]Las estadísticas corresponden al sitio web http://gaia.fdi.ucm.es/projects/jcolibri y fueron obtenidas mediante la herramienta Google Analytics.

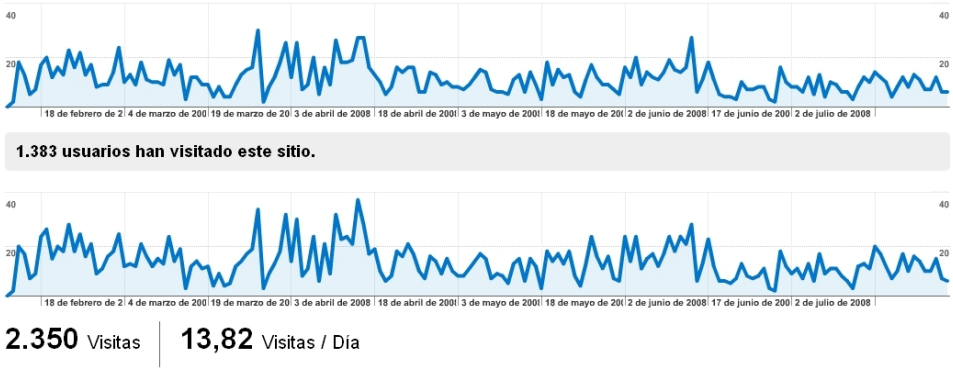


Figura 7.3: Gráfica de usuarios y visitas a la página web de jCOLIBRI

que realizaron más visitas.

Obviamente, España es el país donde se registran más accesos debido a las visitas de los alumnos que utilizaron el armazón para realizar la práctica que se presentará en la próxima sección.

También se tiene constancia de la utilización del armazón tanto a nivel académico como de investigación en las siguientes universidades[6]: Université Pierre et Marie Curie (Paris, Francia), Georgia Institute of Technology (Georgia, USA), Indian Institute of Technology (Chennai, India), The Robert Gordon University (Aberdeen, Reino Unido), University College Cork (Cork, Irlanda), Malardalen University (Vasteras, Suecia), Drexel University (Filadelfia, USA), Federal University of Santa Catarina in Brazil (Santa Catarina, Brasil), y Norwegian University of Science and Technlogy (Trondheim, Noruega).

A nivel empresarial podemos encontrar el sistema KoBaS para la gestión de conocimiento creado a partir de jCOLIBRI por la consultora Wittmann & Partner de Rumanía. La empresa ferroviaria portuguesa EMEF concedió en 2007 el primer premio en I+D a un sistema de diagnóstico inteligente de averías desarrollado en colaboración con la Universidad de Oporto y que estaba implementado a partir de jCOLIBRI. La empresa española MIK también figura como observador tecnológico del armazón y lo han utilizado en sus desarrollos.

Dentro de la propia Universidad Complutense se ha desarrollado un proyecto de final de carrera en colaboración con los servicios centrales de la biblioteca de la UCM para la catalogación automática de documentos (revistas, artículos, tesis, etc.). Este sistema utiliza principalmente los métodos de CBR Textual en jCOLIBRI y ha sido implantado como herramienta en

[6] En el listado se incluyen universidades donde se ha utilizado jCOLIBRI en la realización de prácticas (dentro de cursos oficiales), Master Thesis, Phd Thesis o en proyectos de investigación.

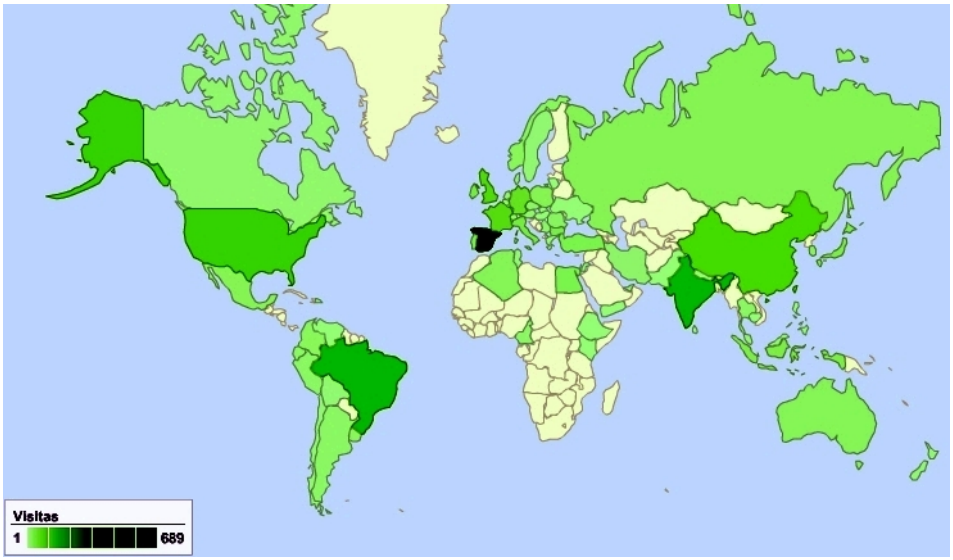


Figura 7.4: Mapa con la distribución de visitas a la página web de jCOLIBRI

dichos servicios.

jCOLIBRI también aparece como herramienta utilizada dentro del proyecto ONE para la creación de un recomendador que ayuda en la conducción de negociaciones. Este proyecto está financiado por el programa "Information Society Technology" de la Comunidad Europea (IST-034744).

7.2 Evaluación por parte de usuarios

En esta memoria se ha presentado jCOLIBRI 2 como una plataforma orientada tanto a usuarios desarrolladores como usuarios diseñadores. Estas dos aproximaciones las hemos denominado *diseño basado en métodos* y *diseño basado en plantillas*. El diseño basado en métodos consiste en programar directamente las aplicaciones CBR a partir del código fuente del armazón, mientras que el diseño basado en plantillas consiste en reutilizar plantillas de sistemas previamente desarrollados y adaptarlas a los requisitos actuales.

Por lo tanto, resulta muy interesante conocer la opinión de los usuarios respecto a ambas aproximaciones. Para obtener dichas opiniones se planteó una práctica a los alumnos del curso Ingeniería de Sistemas Basados en Conocimiento que se imparte como asignatura optativa de segundo ciclo en la Facultad de Informática de la Universidad Complutense de Madrid. Como uno de los objetivos era la comparación de las dos aproximaciones de diseño, la práctica consistió en el desarrollo de un sistema de recomendación ya que por ahora sólo se dispone de plantillas para este tipo de aplicaciones CBR.

	País/territorio	Visitas	Visitas
1.	Spain	689	29,32%
2.	Brazil	198	8,43%
3.	India	197	8,38%
4.	Singapore	194	8,26%
5.	United States	119	5,06%
6.	China	98	4,17%
7.	United Kingdom	82	3,49%
8.	France	77	3,28%
9.	Germany	63	2,68%
10.	Taiwan	44	1,87%

Contribución de País/territorio al total: Visitas

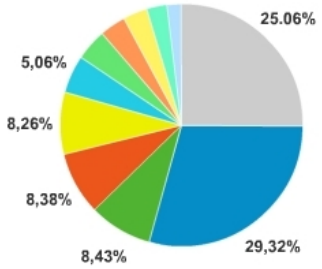


Figura 7.5: Distribución de visitas a la página web de jCOLIBRI

A grandes rasgos, la práctica proponía:

- Diseñar un sistema recomendador.
- Elegir uno de los siguientes procesos de desarrollo: sin utilizar jCOLIBRI 2, mediante el diseño basado en métodos, o mediante el diseño basado en plantillas.
- Finalmente, se debía implementar el recomendador siguiendo el proceso de desarrollo elegido.

Para la práctica, se proporcionó a los alumnos varias bases de datos sobre películas, videojuegos, música,... que podían utilizar en sus recomendadores. La evaluación por parte de los alumnos se obtuvo mediante una serie de encuestas que se realizaron durante la práctica. Los resultados obtenidos sirvieron para comparar las dos aproximaciones de diseño y medir otras variables de control sobre el proceso de desarrollo con jCOLIBRI 2: la cobertura de las platillas respecto a la amplia variedad de diseños propuestos por los alumnos, si la reutilización de las plantillas facilitaba el desarrollo de los sistemas, la "exhaustividad" de los métodos (i.e. qué porcentaje de la funcionalidad a incluir en los recomendadores estaba ya implementada en el armazón), la calidad global de la plataforma y la dificultad de su curva de aprendizaje.

La práctica propuesta a los alumnos se dividía en las siguientes etapas:

- En una primera sesión teórica se expusieron las bases del desarrollo de los sistemas de recomendación mostrando varios ejemplos de aplicaciones reales. Durante esta sesión no se presentó todavía jCOLIBRI ni ninguna de las plantillas incluidas en él (las mostradas en las figuras de la

Sección 6.2.1). Debe destacarse que la mayoría de los sistemas presentados en esta sesión eran recomendadores colaborativos aunque sólo se incluya un sistema de este tipo dentro de la librería de plantillas. Esto nos serviría para medir la posibilidad de mezclar el comportamiento de varias plantillas para implementar los requisitos de un nuevo sistema.

- Después de la primera sesión, los alumnos tuvieron que definir los requisitos y el diseño de sus recomendadores. La única restricción era que dichos sistemas fueran conversacionales. Por lo demás, los alumnos eran libres de definir el dominio, medidas de similitud, interfaz, comportamiento colaborativo o no, y cualquier otra característica.

- A continuación, se presentó jCOLIBRI 2 mediante una clase teórica y una práctica guiada. El ejemplo utilizado fue un sistema básico de recomendación de un único producto (single-shot) utilizando una base de casos sobre viajes. Una vez finalizada la práctica se pasó a los alumnos una primera encuesta para medir sus impresiones iniciales sobre el armazón.

- En la siguiente sesión se presentaron las plantillas de jCOLIBRI 2. Como la herramienta de composición presentada en el Capítulo 6 era sólo un prototipo, se decidió entregar a los alumnos esquemas de las plantillas en papel junto con descripciones de los mismos. Además, se indicaba el código ejecutable correspondiente (que ya estaba incluido en los ejemplos del armazón). Después de esta sesión se pasó una encuesta intermedia para medir la similitud entre los diseños propuestos por los alumnos y las diferentes plantillas presentadas.

- Finalmente se pidió a los alumnos que implementaran su diseño eligiendo entre no utilizar el armazón, el proceso basado en métodos o el proceso basado en plantillas (adaptando el código de los ejemplos substituyendo algunos métodos por otros). Una vez acabada la implementación se realizó otra encuesta donde los alumnos expresaron su opinión final sobre jCOLIBRI.

7.2.1 Resultados de las encuestas

Los 58 alumnos que formaron parte del experimento desarrollaron 24 recomendadores para una amplia variedad de dominios, principalmente: videojuegos (6), películas (5), libros (4), música (3) y relaciones personales (3). Además se implementaron otros pocos recomendadores de pisos, ordenadores e incluso lenguajes de programación.

En general, los resultados de las encuestas fueron muy satisfactorios aunque en ningún momento se obligara a utilizar jCOLIBRI y hubiéramos orientado a los alumnos hacia el desarrollo de sistemas colaborativos cuando sólo

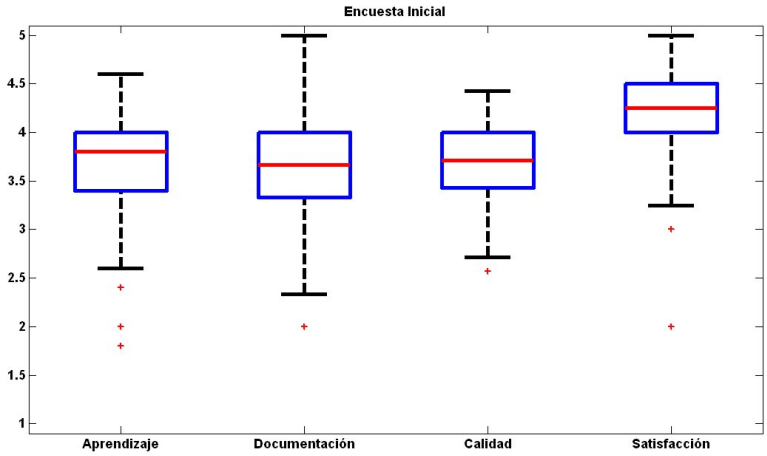


Figura 7.6: Resumen de resultados de la encuesta inicial

se incluye un sistema de este tipo dentro de los ejemplos del armazón. Los resultados detallados de la encuesta se incluyen en el Apéndice A y en este capítulo sólo mostraremos gráficas que resumen dichos resultados.

La encuesta inicial se organizó en cuatro bloques principales: toma de contacto o aprendizaje, calidad de la documentación, calidad del diseño de clases y arquitectura, y nivel de satisfacción general. Los resultados se muestran de forma resumida en la Figura 7.6[7].

En esta encuesta se observa que la curva de aprendizaje no fue pronunciada y que hubo una buena aceptación inicial del armazón debida principalmente a la cantidad y calidad de la documentación y el tutorial. También se puntúa satisfactoriamente la arquitectura de la plataforma, su diseño de clases y ausencia de errores.

La encuesta intermedia sirvió para medir el proceso de desarrollo elegido por los alumnos. Sorprendentemente, todos los usuarios eligieron utilizar jCOLIBRI 2: el 62 % mediante el diseño basado en plantillas y el resto mediante el diseño basado en métodos.

Después de revisar posteriormente los recomendadores implementados, encontramos que no hubo una razón clara para decantarse por una u otra

[7]En este tipo de gráfica, la línea roja indica la posición de la mediana, mientras que los bordes superior e inferior de la caja representa el percentil 25 y 75 (por lo tanto el 50 % de las respuestas se encontraría dentro de la caja azul). Las "asas" indican la posición del valor máximo y mínimo. Si se encuentran datos anómalos se muestran como cruces de color rojo.

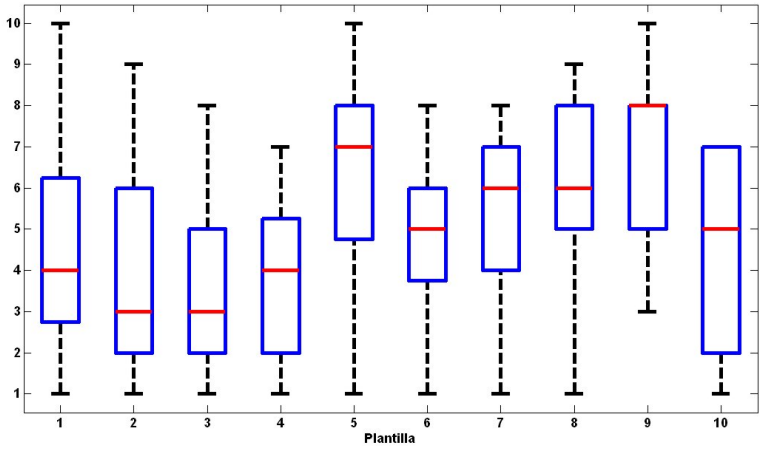


Figura 7.7: Resumen de resultados de la encuesta intermedia

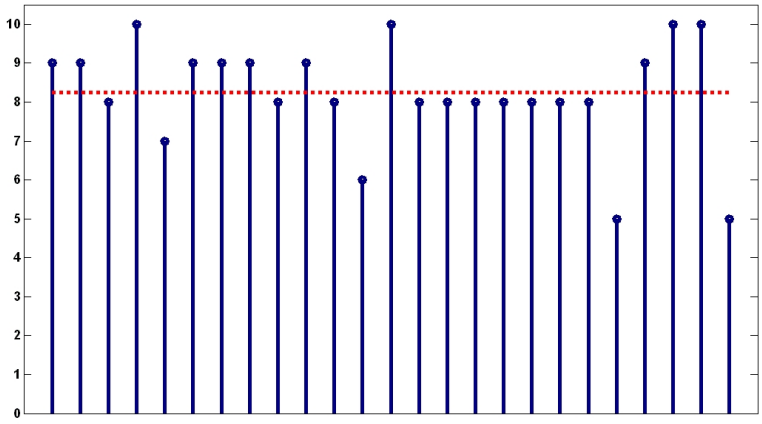


Figura 7.8: Valores máximos asignados a las plantillas

opción. Algunos estudiantes eligieron las plantillas pero realizaron numerosos cambios que desembocaron en sistemas bastante inusuales que probablemente pudieron desarrollarse más fácilmente siguiendo la otra aproximación. En el lado opuesto encontramos varias prácticas implementadas mediante la composición manual de métodos que resultaron ser muy similares a los sistemas de ejemplo de las plantillas. Aún así, pudieron inferirse dos razones que promovieron el diseño basado en métodos. La primera era que algunos alumnos habían realizado diseños iniciales incorrectos (por ejemplo que devolvieran conjuntos de artículos vacíos) y por lo tanto no estaban cubiertos por las plantillas. La otra razón fue la necesidad de incorporar características o métodos especiales que no estaban incluidos en el armazón. Podemos poner como ejemplo un sistema que implementaba el método de recomendación colaborativo de Amazon[8], o un sistema similar a Meetic[9] que requería un filtrado inicial sobre los perfiles de los usuarios.

Aunque los motivos en la elección del proceso de diseño no estuvieran claros, nuestra impresión final después de revisar las prácticas fue que las plantillas cubrían satisfactoriamente los diseños propuestos por los alumnos. Este hecho se corroboraba en la encuesta ya que se planteaban varias preguntas a los alumnos sobre cómo se parecía cada una de las plantillas a su diseño. Los resultados de estas diez preguntas (una por plantilla) se muestran en la Figura 7.7.

Los alumnos que utilizaron las plantillas para implementar el recomendador eligieron la más adecuada a sus requisitos. Por lo tanto, el valor máximo de los asignados a cada plantilla nos indica la utilidad de la mejor plantilla para cada diseño de los alumnos. En media se obtiene un valor de 8.25 sobre 10, lo que indica que la adecuación de las plantillas es bastante alta ya que al menos una de las plantillas era muy similar a su diseño. Estas valoraciones se muestran en el gráfico de la Figura 7.8 donde se refleja el valor máximo asignado por cada alumno y la media de dichas puntuaciones máximas.

La encuesta final sirvió para confirmar la utilidad de las plantillas ya que la mayoría de los alumnos indicaron que simplificaban en gran medida el desarrollo de los recomendadores. Los resultados obtenidos son especialmente importantes si se comparan con una experiencia similar llevada a cabo el curso anterior donde se invitaba a utilizar jCOLIBRI 1 y sólo el 25 % de los alumnos se decidió por esa opción.

Respecto a la completitud de los métodos de jCOLIBRI 2, el 75 % de los alumnos pudo desarrollar sus recomendador sin necesidad de implementar ningún método nuevo. Otro 20 % tuvo que implementar unos pocos y el 5 % bastantes o muchos. Estos resultados significan que nuestro armazón cubre la mayor parte de la funcionalidad necesaria para implementar recomendadores

[8] www.amazon.com
[9] www.meetic.es

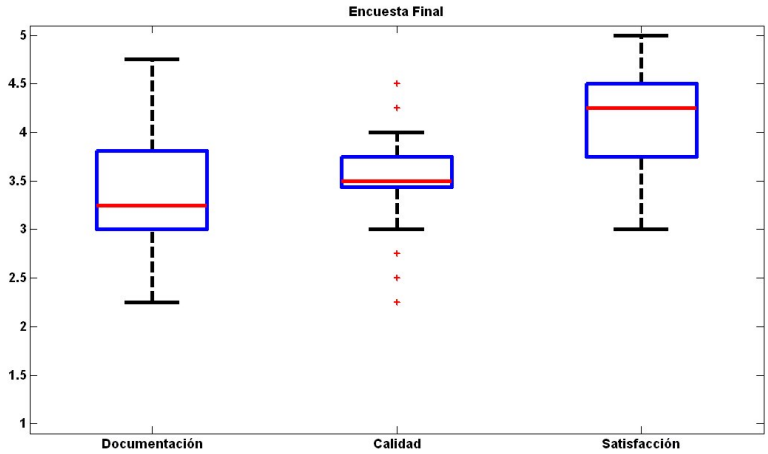


Figura 7.9: Resumen de resultados de la encuesta final

sencillos.

En un segundo bloque de preguntas se repitieron las preguntas de la encuesta inicial sobre la calidad del diseño y la arquitectura del armazón para poder comparar los resultados. También se repiten las preguntas sobre la documentación y sobre la satisfacción general final. Como se resume en la gráfica de la Figura 7.9, los resultados obtenidos son tan buenos como los de la encuesta inicial aunque la complejidad del recomendador implementado fuera mucho mayor que en la práctica guiada que sirvió como criterio para la primera encuesta.

7.3 Conclusiones

En este capítulo se presentan una serie de datos sobre el uso de jCOLIBRI por la comunidad CBR y sobre la opinión de los usuarios respecto a la calidad y utilidad del mismo.

Los datos de uso de jCOLIBRI y las visitas a la página web sirven para hacer una estimación informal de la repercusión del armazón a nivel internacional. El alto número de descargas implica una gran difusión y utilización, mientras que la distribución geográfica de las visitas a la página web demuestra un amplio interés en la comunidad CBR.

Además se presentan una serie de resultados obtenidos a partir de encuestas realizadas a un grupo de alumnos que utilizó el armazón en la implementación de un recomendador. Dichas encuestas sirvieron para medir la opinión de estos alumnos sobre aspectos como la calidad, documentación y facilidad de desarrollo. Los resultados obtenidos fueron muy satisfactorios

y también sirvieron para comparar las dos capas del armazón: la dirigida a desarrolladores y la dirigida a diseñadores. Para ello se pidió a los alumnos que eligieran entre el diseño basado en métodos y el diseño basado en plantillas, obteniéndose que el 62 % de los estudiantes prefirieron la utilización de las plantillas. Respecto a la cobertura de los métodos implementados, se obtuvo que el 75 % no necesitó implementar ningún método que no estuviera incluido en el armazón para desarrollar su recomendador. Los resultados detallados de las encuestas se han incluido en el Apéndice A.

Capítulo 8

Conclusiones y Trabajo Futuro

– ¡Terminé! Al final he acabado con toda una caja de herramientas. Pero bueno, era la mejor forma de conseguir un buen martillo.

Los resultados obtenidos en este trabajo de tesis se enmarcan dentro de la plataforma jCOLIBRI. Entre ellos podemos destacar una arquitectura de referencia para el desarrollo aplicaciones CBR, la implementación de dicha arquitectura en una completa plataforma que incluye e integra diversos métodos CBR, y por último, un nuevo mecanismo de composición de aplicaciones CBR basado en la representación y razonamiento con plantillas reutilizables. Finalmente se ha realizado una evaluación del trabajo de tesis desde el punto de vista de su repercusión en la comunidad CBR y de la opinión de los usuarios que utilizan jCOLIBRI.

A continuación pasaremos a exponer las principales conclusiones, aportaciones y limitaciones de dicho trabajo así como posibles líneas de continuación del mismo.

8.1 Conclusiones

Son varias las conclusiones obtenidas a partir de este trabajo de tesis. La principal es la utilidad de contar con una plataforma como jCOLIBRI para el desarrollo de sistemas CBR tanto desde el punto de vista académico como de investigación. Una característica muy importante de la plataforma es su orientación tanto a usuarios desarrolladores como diseñadores. Aunque ambos roles estén muy diferenciados, en jCOLIBRI se demuestra que es posible integrar en una única plataforma la funcionalidad y herramientas necesarias para cubrir las necesidades de estos dos tipos de usuario. Para ello se propone una arquitectura multi-nivel donde se ofrece una capa específica para los usuarios desarrolladores y otra capa para los diseñadores. A continuación,

detallaremos como han ido evolucionando los conceptos e ideas de diseño de la plataforma hasta alcanzar esta arquitectura multi-nivel que permite tanto construir como generar aplicaciones CBR.

Inicialmente jCOLIBRI se planteó según la metodología KADS para el diseño de sistemas expertos. Esta metodología propone la representación del comportamiento del sistema como una progresiva descomposición en tareas que son finalmente resueltas por métodos que implementan la funcionalidad necesaria. De esta forma, la primera versión de la plataforma –jCOLIBRI 1– ofrecía un completo interfaz gráfico que permitía configurar los sistemas CBR sin apenas necesidad de programación (Recio-García et al., 2005c; Díaz-Agudo et al., 2007a). Sin embargo, esta versión inicial presentaba ciertos inconvenientes que se derivaban principalmente del acoplamiento entre el código de los métodos CBR y las herramientas de composición visuales. Esto limitaba su extensibilidad y proyección futura, por lo que se planteó la evolución de la plataforma siguiendo el diseño de una nueva arquitectura donde se tuvieran en cuenta los errores y aciertos de su predecesora (Recio-García et al., 2006b).

La premisa más importante en el diseño de la nueva arquitectura fue que una plataforma como jCOLIBRI debía diferenciar claramente los tipos de usuario que la utilizarían y el tipo de armazón requerido por cada perfil. Al igual que en su predecesora, jCOLIBRI 2 pretendía servir tanto a usuarios desarrolladores como diseñadores, lo que suponía distintos requisitos sobre la plataforma. Los usuarios desarrolladores son aquellos que quieren programar directamente sobre el código y buscan un armazón de caja blanca. Los armazones de caja blanca se basan principalmente en la aplicación de mecanismos de extensión y flexibilización. Sin embargo, los usuarios diseñadores prefieren seleccionar algunos de los componentes predefinidos del armazón, parametrizarlos y conectarlos. Los armazones de este tipo se denominan de caja negra, y son más sencillos de comprender y los que menor esfuerzo suponen al usuario. En cambio, son menos flexibles que los anteriores y los más complejos de diseñar. La evolución natural de un armazón comienza con un diseño de caja blanca que desemboca en una arquitectura de caja negra.

Esta diferenciación entre armazones de caja blanca que se completan con herramientas de composición de caja negra, junto con los problemas de acoplamiento de código de jCOLIBRI 1 desembocaron en el diseño multi-nivel de la arquitectura e implementación de jCOLIBRI 2. En esta nueva arquitectura se separan las dos capas de la plataforma: una orientada a desarrolladores y otra orientada a diseñadores.

La capa para desarrolladores de jCOLIBRI 2 incluye los componentes básicos para implementar sistemas CBR: mecanismos de persistencia que abstraen del medio concreto utilizado, representación y organización en memoria

de los casos, etcétera. Por lo tanto, esta parte de la plataforma puede definirse como un *armazón* que puede ser extendido y reutilizado por los usuarios desarrolladores para construir sistemas CBR. Una característica a resaltar en este armazón es que se aprovecha de las nuevas tecnologías Java para facilitan el desarrollo tanto de aplicaciones independientes como con interfaz web. Sin embargo, la aportación más importante de esta capa es que ofrece el marco integrador necesario para incorporar métodos y técnicas CBR de distinta naturaleza. Esto se demuestra mediante la inclusión de técnicas comunes de CBR más otras dos grandes extensiones con métodos de CBR Textual y CBR con conocimiento intensivo. La primera extensión se encarga de manipular casos representados en forma de textos y la segunda permite reutilizar conocimiento adicional sobre el dominio entre distintas aplicaciones si dicho conocimiento está representado mediante ontologías.

Dentro de la extensión de CBR Textual de jCOLIBRI se incluyen dos tipos de métodos. El primer tipo de métodos –denominados *semánticos*– intenta capturar el significado del texto y estructurarlo dentro de un caso que pueda ser manejado mediante técnicas genéricas de CBR (Recio-García et al., 2005a). Sin embargo, el segundo tipo de métodos opta por aplicar algoritmos *estadísticos* muy conocidos y probados en el área de Recuperación de Información (Recio-García et al., 2007c).

Los métodos de CBR con conocimiento intensivo parten de la premisa de que las ontologías permiten desacoplar el conocimiento adicional que necesitan muchos sistemas CBR y reutilizarlo entre aplicaciones. Esto hace posible desarrollar métodos genéricos reutilizables que puedan ser incluidos en una plataforma como jCOLIBRI. Esta idea queda contrastada en Recio-García et al. (2006a); Díaz-Agudo et al. (2008b) donde se propone el uso de ontologías como medio de persistencia, para calcular la similitud entre atributos y para aumentar la expresividad de las consultas del usuario. En este último caso, también es posible combinar el uso de ontologías con los métodos de CBR Textual para procesar consultas en lenguaje natural (Díaz-Agudo et al., 2007b). Otra gran aportación de las ontologías al ciclo CBR la encontramos en la etapa de adaptación ya que permiten reutilizar el conocimiento del dominio necesario para adaptar las soluciones (Díaz-Agudo et al., 2008b).

El otro nivel de la plataforma está orientado a usuarios diseñadores e incluye herramientas que facilitan el diseño y generación semiautomática de aplicaciones CBR mediante la composición de los métodos y componentes del armazón de la capa para desarrolladores. En este nivel de la plataforma se presenta una novedosa aproximación basada en *plantillas* que abstraen el comportamiento de los sistemas CBR y que pueden ser instanciadas con los distintos componentes de la plataforma (Recio-García et al., 2009). Este *diseño basado en plantillas* propone aplicar el mismo paradigma CBR de reutilización de experiencias al campo de la composición de software. Aquí

los casos consisten en composiciones previas de sistemas CBR que son recuperadas y adaptadas, es decir: utilizar CBR para generar sistemas CBR. Los mecanismos y técnicas necesarias para recuperar plantillas y adaptarlas a las necesidades del diseñador CBR fueron uno de los retos de investigación más importantes en este trabajo. Además, las soluciones a dichos retos que se presentan en esta memoria ponen de manifiesto la utilidad de los componentes de jCOLIBRI junto con las aportaciones de la Web Semántica. Esta última característica se debe a que en el desarrollo de esta capa de la plataforma se decidió utilizar las tecnologías y estándares de los Servicios Web Semánticos (SWS) en vez de la aproximación KADS. Las principales razones de este cambio fueron dos: primero que los SWS son en cierta forma una evolución de la metodología KADS, y segundo porque la composición semiautomática de los sistemas CBR requiere ciertas capacidades de razonamiento sobre los componentes que se encuentran disponibles en los estándares de SWS ya que están basados en representaciones mediante ontologías.

Las plantillas están formadas por un flujo de tareas donde se permiten elementos de control más complejos que las secuencias utilizadas en jCOLIBRI 1: bucles, condicionales, etc. Este tipo de representación se define en una ontología denominada *CBROnto* en la que se adaptan los estándares de los Servicios Web Semánticos. El uso de CBROnto como mecanismo de representación de las plantillas permite razonar sobre su estructura y características. Esta capacidad de razonamiento se aprovechará en los procesos de recuperación y adaptación que se describen a continuación.

El mecanismo de recuperación de plantillas constituye una aportación muy significativa del trabajo de tesis, ya que no es posible utilizar un lenguaje de consulta debido a la complejidad del dominio y a su difícil aceptación por parte del usuario. En su lugar se presenta una aproximación que reutiliza los métodos para la implementación de sistemas de recomendación que también incluye el armazón (Recio-García et al., 2008c). Estos métodos son utilizados normalmente para recomendar artículos como libros, aparatos electrónicos, e incluso contactos personales, pero no habían sido aplicados anteriormente a la recomendación de composiciones software –como lo son las plantillas. El proceso de recuperación planteado es un sistema de recomendación mediante propuestas que muestra diversas composiciones al usuario hasta que elige la más apropiada (Recio-García et al., 2008a). Esto presenta el inconveniente de encontrar un mecanismo que permita al usuario comprender el funcionamiento de las diferentes propuestas. Como solución se ha desarrollado una aproximación bautizada como *Recuperación Probando* donde se permite que el usuario ejecute realmente la composición software que le está siendo propuesta. El aspecto más importante en la implementación de este tipo de recomendadores es la medida de similitud que permite comparar las plantillas. Por ello, la medida de similitud desarrollada compara la estructura y métodos utilizados en cada plantilla. Además, esta medida de similitud

aprovecha que las plantillas estén representadas mediante CBROnto para utilizar información semántica adicional sobre ellas.

Una vez elegida la composición más similar se pasa a la etapa de adaptación. Esta etapa del diseño basado en plantillas también supuso un reto de investigación dentro del trabajo de tesis. En el proceso de adaptación presentado, el diseñador asigna o modifica los métodos que instancian las distintas tareas que forman la plantilla. Estos métodos son los mismos que utilizan los desarrolladores que programan directamente sobre código y que se encuentran disponibles en la otra capa de la plataforma. Por lo tanto, para que el sistema pueda razonar con ellos es necesario añadirles información semántica sobre su funcionamiento y requisitos de ejecución. De esta forma, la herramienta de composición podrá indicar y sugerir al usuario qué métodos son los más adecuados en la situación actual para resolver una tarea. La información semántica de los métodos es el punto de enlace entre las dos capas de la plataforma jCOLIBRI y se obtiene mediante anotaciones basadas en los términos de la ontología CBROnto. Esta ontología contiene los conceptos necesarios para representar el funcionamiento de los métodos y, una vez más, esta representación se toma de los estándares de los Servicios Web Semánticos. Para enlazar entre los métodos Java de capa para desarrolladores del armazón y la representación semántica mediante CBROnto que utilizan las herramientas de composición de la capa para diseñadores se utilizan *anotaciones*. Este mecanismo permite añadir meta-información en el código fuente de forma elegante, transparente, y que a su vez sirve como documentación.

La información semántica añadida a los métodos consiste principalmente en sus pre-/postcondiciones y la tarea que resuelven. Las pre- y postcondiciones se basan en los parámetros de entrada y salida del método. Estos parámetros no son directamente los tipos Java sino que se definen también mediante una jerarquía de conceptos de CBROnto. Esta jerarquía de tipos en CBROnto permite que la herramienta de composición pueda definir si el flujo de datos entre métodos es o no compatible. Para ello se utilizan los mecanismos de razonamiento de las Lógicas Descriptivas. Además, la representación de los métodos elegida ofrece la posibilidad de indicar condiciones que no son obligatorias, sino que actúan como recomendaciones de diseño. Estas recomendaciones también serán utilizadas por la herramienta de composición para sugerir ciertos métodos al usuario.

La representación semántica de los métodos permite que la herramienta de composición guie al usuario en la asignación de métodos a las tareas y en la configuración del flujo de datos entre los parámetros de los distintos métodos. Además, se encarga de comprobar en todo momento que el diseño realizado es correcto. Una vez finalizada la composición de la aplicación CBR a través la plantilla, la herramienta permite generar automáticamente el código de dicho sistema CBR ya que dispone de toda la información necesaria gracias

a la representación semántica de las plantillas y los métodos.

8.1.1 Repercusión del trabajo de tesis

El Capítulo 7 de la memoria presenta una evaluación del trabajo de tesis desde el punto de vista cuantitativo y el cualitativo. La evaluación cuantitativa de jCOLIBRI se basa en el número de descargas del mismo –lo que indica el número de usuarios potenciales–, el número de visitas a la página web, y la utilización del armazón a a nivel académico, de investigación y empresarial. Los datos recopilados muestran una media aproximada de 200 descargas al mes de jCOLIBRI 2, alcanzando prácticamente un total de 5.000 descargas entre ambas versiones del armazón. Además, se registraron unas 2.350 visitas a la página web del proyecto por parte de casi 1.400 usuarios de 71 países distintos durante la primera mitad de 2008. Por su parte, OntoBridge –subproyecto de jCOLIBRI– acumula más de 750 descargas desde mayo de 2007.

Respecto a la repercusión del armazón a nivel académico y de investigación, se tiene constancia de la utilización de jCOLIBRI en una decena de universidades internacionales. También existen varios sistemas empresariales y un proyecto de investigación europeo que lo integran dentro de sus desarrollos.

La evaluación desde el punto de vista cualitativo se realizó por medio de una serie de encuestas a los alumnos de un curso del segundo ciclo de Ingeniería Informática en la Universidad Complutense de Madrid. Los alumnos tuvieron que implementar un sistema recomendador como una de sus prácticas y se les propuso utilizar jCOLIBRI como herramienta de desarrollo. Aunque el uso del armazón no fuera obligatorio todos los alumnos decidieron utilizarlo. A la hora de elegir entre la programación directa utilizando la parte de caja-blanca (que denominamos *diseño basado en métodos*) y el desarrollo mediante plantillas (*diseño basado en plantillas*), el resultado fue de un 62 % decantandose por esta última opción. En general, la adecuación de las plantillas para desarrollar sus recomendadores fue bastante alta (8.5 sobre 10) y los métodos de jCOLIBRI cubrieron en un 75 % la funcionalidad necesaria para la implementación. La documentación, calidad y satisfacción global del armazón alcanzaron respectivamente una nota media de 3.2, 3.5, 4.3 sobre una escala de 5.

8.2 Aportaciones

A partir de las conclusiones presentadas en la sección anterior podemos desglosar las aportaciones de este trabajo de tesis en:

- Definición y desarrollo de una plataforma multi-nivel para la construcción y generación de sistemas CBR orientada tanto a usuarios desarrolladores como diseñadores. Esta plataforma ofrece un marco integrador de las técnicas y métodos que se produzcan en la comunidad CBR.

- Dentro del nivel de la plataforma orientado a usuarios desarrolladores se incluye:

  - Una arquitectura para el desarrollo de sistemas CBR que permite la integración de métodos CBR de diversa naturaleza. Para ello se se especifican y definen los elementos centrales de la arquitectura que sirven para estructurar la construcción de aplicaciones CBR: separación del medio de persistencia, representación de los casos, organización de la base de casos, etc.
  - La implementación de dicha arquitectura en una plataforma donde se dispone de los componentes necesarios para desarrollar una amplia variedad de sistemas CBR.
  - Métodos reutilizables para implementar aplicaciones de CBR Textual. Dentro de este grupo de métodos se distingue y discute la aplicabilidad de los *métodos semánticos* que obtienen una representación estructurada del texto, y los *métodos estadísticos* que aplican técnicas de Recuperación de Información.
  - Métodos reutilizables para implementar aplicaciones de CBR con conocimiento intensivo. En dicho desarrollo se demuestra la viabilidad del uso de ontologías para encapsular el conocimiento adicional del sistema CBR. En los métodos propuestos, las ontologías se sirven de medio de persistencia, para definir las medidas de similitud y para guiar el proceso de adaptación.
  - Una metodología para procesar consultas del usuario en lenguaje natural donde se combinan los métodos de CBR Textual y CBR con conocimiento intensivo.
  - Una metodología para obtener representaciones de los casos mediante ontologías. Para ello se aplican los métodos de CBR Textual incluidos en el armazón combinados con técnicas de Análisis Formal de Conceptos.

- Respecto a la parte de la plataforma orientada a usuarios diseñadores, se ha desarrollado un mecanismo para la composición semiautomática de sistemas CBR basada en plantillas donde:

  - Se utilizan los estándares de la Web Semántica, más concretamente del campo de los Servicios Web Semánticos, para representar el flujo de ejecución, el flujo de datos y el comportamiento de los métodos de jCOLIBRI.

- Se desarrolla una aproximación basada en plantillas para el diseño de sistemas CBR. Las plantillas representan abstracciones de sistemas CBR y el usuario recupera la más similar para adaptarla a sus necesidades. De este modo, el mecanismo propuesto sigue también una aproximación CBR donde las plantillas forman la base de casos.
- Se define un lenguaje de representación de las plantillas mediante la ontología CBROnto. Esta representación es una adaptación de los estándares de los Servicios Web Semánticos.
- Se desarrolla un mecanismo de recuperación de plantillas mediante un sistema de recomendación. Este sistema de recomendación utiliza una novedosa aproximación que permite al usuario ejecutar las composiciones recomendadas para formarse una opinión sobre las mismas. Este mecanismo se bautiza como *Recuperar Probando*.
- Se investiga la función de similitud óptima que permita implementar el sistema de recomendación. Para ello se combinan medidas basadas en la estructura y funcionalidad de las plantillas, además de cierta información semántica que también se define en CBROnto. Esta información permite que el experto en el dominio indique de forma más precisa cuales son las características a comparar entre plantillas.
- Se establecen los requisitos de representación semántica para los componentes de jCOLIBRI y se desarrolla un lenguaje mediante CBROnto adaptando de nuevo los estándares de los SWS y que se complementa con la representación de las plantillas.
- Se define un mecanismo para enlazar las dos capas de jCOLIBRI mediante anotaciones en el código fuente que permiten añadir la información semántica de forma transparente.
- Se definen los elementos auxiliares para representar flujos de datos dentro de las plantillas mediante variables intermedias.
- A partir de las representaciones semánticas de las plantillas y los componentes del armazón se desarrolla un mecanismo de composición semiautomática que utiliza dicho conocimiento para ayudar al usuario en el proceso de diseño de aplicaciones CBR.

8.3 Limitaciones

La complejidad inherente de desarrollar una plataforma genérica para el desarrollo de cualquier tipo de sistema CBR, hace que que sean varias las limitaciones encontradas. Para analizarlas seguiremos también la división en capas de jCOLIBRI.

La plataforma actual jCOLIBRI 2 es el fruto de una evolución durante el trabajo de tesis donde se resuelven los problemas surgidos en las versiones anteriores. Dentro de la arquitectura definitiva, la capa que más se beneficia de esta evolución es la orientada a desarrolladores. El diseño de esta capa se basó en la reutilización de los elementos de la arquitectura de jCOLIBRI 1 que demostraron ser acertados, resolviendo a su vez los problemas encontrados en dicha versión inicial. Por lo tanto, podemos decir que la capa de desarrolladores carece de limitaciones significativas gracias a esa evolución donde se resuelven los inconvenientes aparecidos previamente. Al revisar el listado de limitaciones de jCOLIBRI 1 (Sección 3.1.3, pág. 86) encontramos que todas las limitaciones se resuelven en la versión posterior excepto una: un lenguaje de consultas más expresivo. Sin embargo, con la incorporación de los métodos de sistemas CBR de recomendación se resuelve este problema de cierta manera, ya que dichos métodos están orientados a mejorar la interacción con el usuario para obtener sus preferencias. Además, en la explicación del mecanismo de persistencia de jCOLIBRI 2 se plantea como posible solución el uso del lenguaje HQL (Hibernate Query Language, ver Bauer y King (2006)) para expresar las consultas sobre casos almacenados en bases de datos.

La parte orientada a diseñadores de jCOLIBRI 2 sí que presenta algunas limitaciones reseñables debido a la complejidad de los retos de investigación abordados. Desde un punto de vista conceptual encontramos que el mecanismo de composición de métodos se basa en una simplificación donde se asume que las pre-/postcondiciones de los métodos están definidas únicamente por sus entradas y salidas. Aunque esta afirmación se acompaña de una completa justificación, el mecanismo planteado podría ampliarse para incluir otro tipo de restricciones lógicas a las pre-/postcondiciones de los métodos. De cualquier manera, hay que recordar que la representación de los métodos mediante CBROnto incluye los elementos necesarios para incluir esta información extra de una forma sencilla.

Otra de las limitaciones de la plataforma podríamos encontrarla al intentar ampliar el sistema de plantillas para otro tipo de aplicaciones CBR. Más concretamente creemos que los puntos más sensibles de esta aproximación son: el cálculo de similitud, ya que está muy ligado a la librería de plantillas utilizada, y jerarquía de tipos de CBROnto, porque depende de los parámetros de los métodos a componer. Como solución a estos problemas, en la próxima sección de trabajo futuro se presenta una propuesta que paliaría las limitaciones anteriores. A pesar de ello, nuestra opinión sigue siendo que el el diseño basado en plantillas puede aplicarse a casi cualquier tipo de sistemas CBR. Por lo tanto, esta afirmación debe ser demostrada incluyendo plantillas de otro tipo de aplicaciones CBR. Aunque no se hayan incluido en la memoria, se dispone de una serie de plantillas para sistemas de CBR Tex-

tual que todavía no han sido incorporadas a la herramienta y que validarían parcialmente la premisa anterior.

La representación de las plantillas y métodos a través de CBROnto no incluye los elementos necesarios para almacenar el flujo de datos que se define al realizar la composición. Esto se debe a que actualmente sólo se dispone de un prototipo de la herramienta de composición que no incluye capacidades de almacenamiento de las plantillas instanciadas. Esta característica no se ha desarrollado todavía ya que no es necesaria para comprobar la validez del diseño basado en plantillas propuesto en esta memoria. Sin embargo, debe ser incluida en a corto plazo para completar la herramienta.

Por último, el objetivo de la herramienta de composición de sistemas CBR es generar el código de una aplicación semi-completa que sea completada por el usuario. Esto requiere de la funcionalidad necesaria para editar el código fuente, encontrar sus errores, compilarlo y ejecutarlo. La herramienta incluye esta funcionalidad de forma muy básica, pero en algunos ocasiones donde el sistema CBR estuviera combinado con aplicaciones más grandes, sería necesario disponer de capacidades de edición más potentes.

8.4 Trabajo futuro

Las limitaciones presentadas en la sección anterior nos sirven como base para indicar el trabajo futuro:

- Definir e implementar un mecanismo de definición de consultas genérico y expresivo.
- Extender CBROnto para permitir la representación del flujo de datos y así poder almacenar en la base de casos de la herramienta de composición nuevos sistemas que haya configurado el usuario.
- Incluir plantillas de otro tipo de sistemas CBR en la herramienta de composición. A corto plazo se añadirán las plantillas de CBR Textual.
- Debe comprobarse que el proceso de recuperación sigue funcionando correctamente al ampliar la base de casos. Según se comentaba en el apartado de limitaciones, la medida de similitud utilizada en la recuperación está muy ligada a la librería de plantillas actual. Por lo tanto sería necesario ampliar esta medida de similitud para que fuera independiente de las plantillas que haya en la base de casos. Un posible forma de ampliar la función de similitud consiste en incorporar el flujo de datos entre métodos como factor que determina el parecido entre plantillas.
- Según se amplíe la librería de plantillas, sería necesario comprobar si el mecanismo para definir las pre-/postcondiciones de los métodos a tra-

vés de la jerarquía de tipos de CBROnto es lo suficientemente expresivo, y si debe ser extendido. Una posible línea de trabajo futuro consistiría en obtener esa jerarquía de tipos de forma automática a partir de los métodos a componer. Esta jerarquía extraída automáticamente serviría como base al diseñador para crear nuevos tipos –conceptos– que expresen restricciones.

- Una vez que se dispusiera de la generación automática de la jerarquía de conceptos, podría simplificarse enormemente el proceso de anotación de los métodos en el código fuente. En la memoria se propone que el desarrollador cree estas anotaciones de forma manual, sin embargo, a excepción de la tarea a resolver, el resto de la información podría extraerse automáticamente –siempre que se dispusiera de la jerarquía de conceptos correspondiente. Por lo tanto, una futura línea de trabajo podría explorar esta posibilidad para automatizar aún más el proceso de composición.

- Ampliar las capacidades de edición y compilación del código generado por la herramienta de composición. Para ello, se pretende incorporar las herramientas desarrolladas dentro del editor Eclipse[1].

- Uno de los elementos centrales de la plataforma jCOLIBRI 2 es la utilización de los estándares de los Servicios Web Semánticos. Según se menciona a lo largo de la memoria, la adopción de este tipo de representación posibilita un futuro uso descentralizado de métodos que no se ejecutaran localmente sino como servicios remotos. Por lo tanto, otra de las líneas de trabajo futuro consistiría en permitir la ejecución remota de los métodos CBR de forma que cada desarrollador hiciera públicos sus métodos como servicios web y jCOLIBRI se encargara de componerlos y ejecutarlos.

Aunque una de las necesidades del armazón es la inclusión de nuevos métodos, creemos que dicha tarea debe distribuirse sobre la comunidad de expertos CBR. El objetivo de jCOLIBRI es servir como marco integrador de los métodos desarrollados por los investigadores del área de CBR, y por lo tanto, el trabajo futuro no debería centrarse en la implementación de los métodos teóricos diseñados por terceros, sino en el soporte a los mismos. Esto no implica que no se realice investigación sobre nuevos métodos CBR dentro de jCOLIBRI, sino todo lo contrario, ya que ahora disponemos de la plataforma necesaria para desarrollar nuevas técnicas CBR. De cualquier modo, el trabajo futuro deberá siempre orientarse para conseguir nuestro objetivo de convertir jCOLIBRI en la plataforma de referencia para la comunidad CBR.

[1] http://www.eclipse.org

Apéndice A

Encuestas

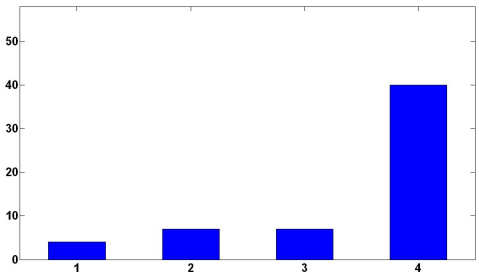
A continuación se incluyen las encuestas realizadas a los alumnos para la evaluación de jCOLIBRI 2 y cuyo planteamiento y resultado se presentó en la Sección 7.2.

A.1 Encuesta Inicial

La primera encuesta se realizó a 58 alumnos después de realizar una primera práctica guiada con jCOLIBRI. Dicha práctica era un simple recomendador (single-shot) de viajes creado a partir del ejemplo que aparece en el tutorial.

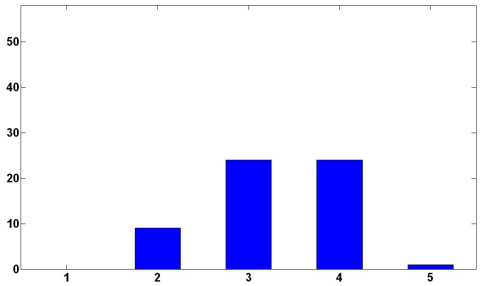
Pregunta 1. ¿Cómo fue tu primera toma de contacto con jCOLIBRI en el ejemplo de los viajes?

1. No me enteraba de qué iban los ejemplos.
2. No me enteraba de cómo empezar a crear una aplicación CBR con jCOLIBRI.
3. Tuve problemas de instalación.
4. No tuve problemas.



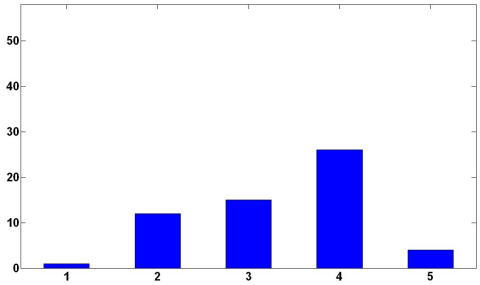
Pregunta 2. ¿Te ha costado aprender el funcionamiento de jCOLIBRI?

1. Mucho.
2. Bastante.
3. Regular.
4. Poco.
5. Nada.



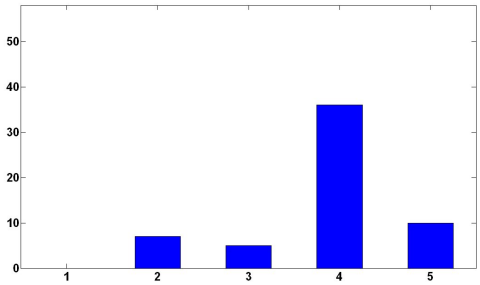
Pregunta 3. ¿Te ha ayudado el tutorial que hay en la página web para aprender el funcionamiento de jCOLIBRI?

1. Nada.
2. Poco.
3. No lo he leído.
4. Bastante.
5. Mucho.



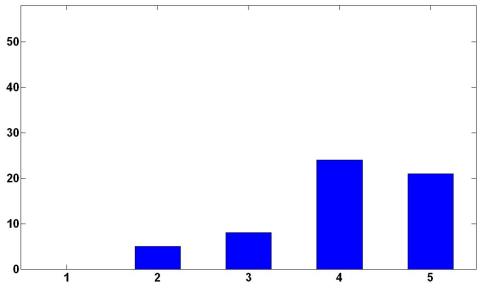
Pregunta 4. ¿Te ha parecido que el enunciado de la práctica es claro y fácil de seguir?

1. Nada.
2. Poco.
3. Regular.
4. Bastante.
5. Mucho.



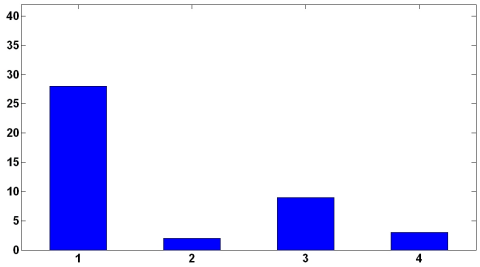
Pregunta 5. ¿Te han sido útiles los ejemplos de código java incluidos en el framework?

1. Nada.
2. Poco.
3. No los he leido.
4. Bastante.
5. Mucho.



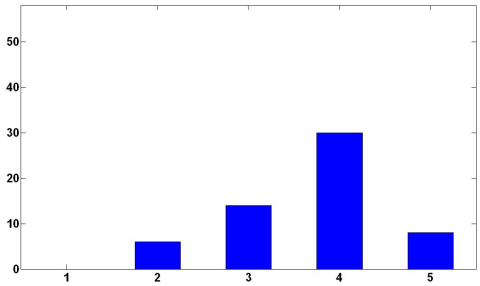
Pregunta 6. ¿Se te ocurre alguna forma de facilitar el aprendizaje de jCOLIBRI? (Respuesta libre)

1. No, así es correcto.
2. Detallando más el ejemplo.
3. Con más ejemplos que muestren otra funcionalidad.
4. Traduciendo al español.



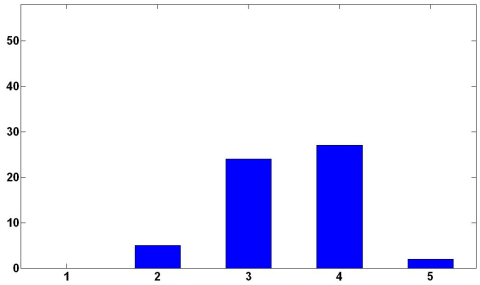
Pregunta 7. ¿Cómo de completa te ha parecido la documentación de jCOLIBRI (tutoriales, ejemplos y API)?

1. Muy escueta.
2. Escueta.
3. Suficiente.
4. Bastante completa.
5. Muy completa.



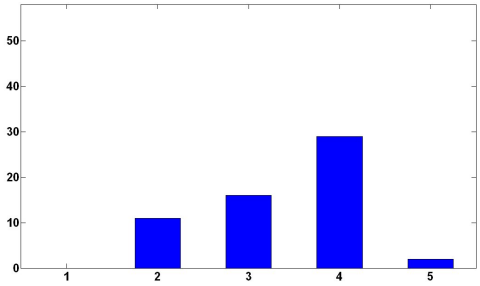
Pregunta 8. ¿Cómo evaluarías la facilidad para encontrar información dentro la documentación de jCOLIBRI (tutoriales, ejemplos y API)?

1. No se encuentra nada.
2. Se encuentra pero cuesta un poco.
3. Normal.
4. Se encuentra bastante bien.
5. Es muy fácil encontrar lo que buscas.



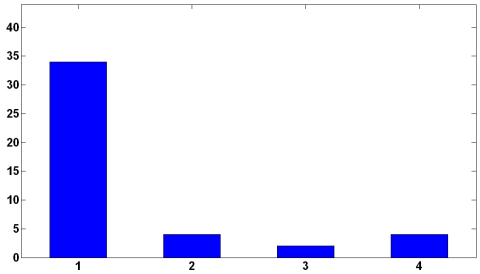
Pregunta 9. ¿Cómo evaluarías la claridad en las explicaciones de la documentación de jCOLIBRI (tutoriales, ejemplos y API)?

1. No se entiende nada.
2. Se entiende con dificultad.
3. Normal.
4. Se entiende bastante bien.
5. Está todo muy bien explicado.



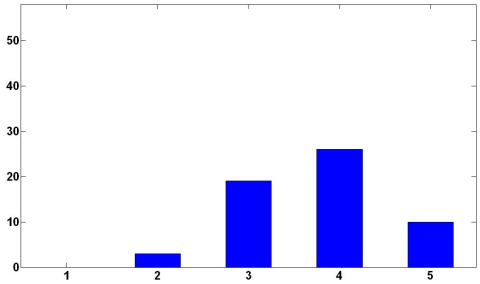
Pregunta 10. ¿Añadirías algo más a la documentación? (Respuesta libre)

1. No, así está bien.
2. Traducirla al castellano.
3. Incluir referencias a documentación de librerías externas como Hibernate o Java Beans.
4. Otros.



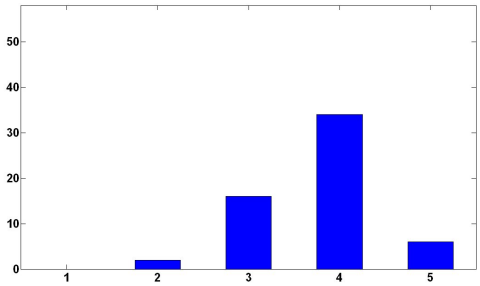
Pregunta 11. ¿Qué te parece la arquitectura de jCOLIBRI: capa de conectores, organización en memoria y métodos?

1. Muy mala.
2. Mala.
3. Regular o no la conozco.
4. Buena.
5. Muy buena.



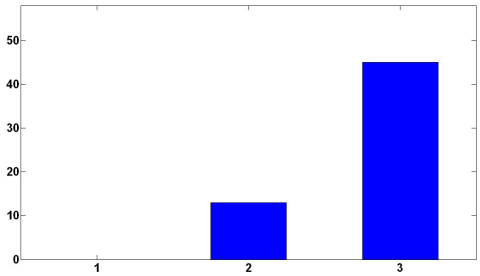
Pregunta 12. ¿Qué te parece el diseño de clases de jCOLIBRI?

1. Muy malo.
2. Malo.
3. Regular o no lo conozco.
4. Bueno.
5. Muy bueno.



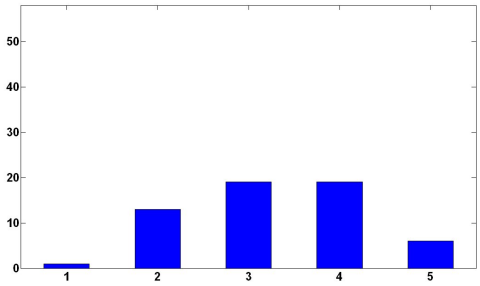
Pregunta 13. ¿Has encontrado algún fallo/bug en el código?

1. Fallos graves.
2. Algún fallo sin demasiada importancia.
3. Ninguno.



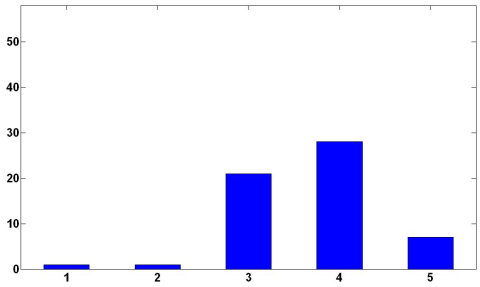
Pregunta 14. ¿Qué te parece la configuración y el uso de los conectores?

1. La configuración y uso es muy complejo.
2. La configuración es compleja pero una vez aprendida parece más fácil.
3. Regular o no tengo experiencia suficiente para opinar.
4. Son fáciles de usar una vez configurados.
5. Son fáciles de usar y configurar.



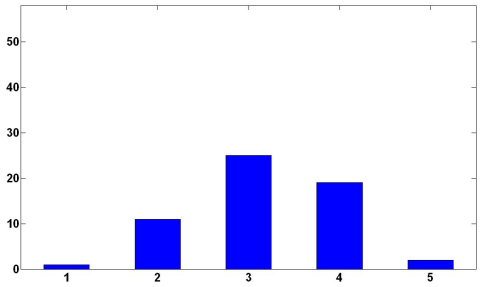
Pregunta 15. ¿Crees que jCOLIBRI facilita la creación de aplicaciones CBR con ontologías?

1. Nada.
2. Poco.
3. Regular o no tengo experiencia suficiente para opinar.
4. Bastante.
5. Mucho.



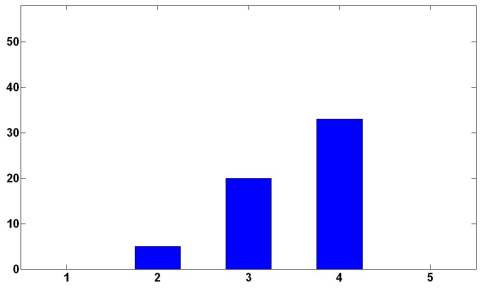
Pregunta 16. ¿Es fácil depurar las aplicaciones en jCOLIBRI?

1. Nada.
2. Poco.
3. Regular.
4. Bastante.
5. Mucho.



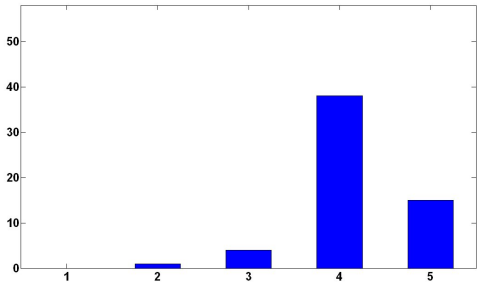
Pregunta 17. ¿Crees que jCOLIBRI es eficiente?

1. Nada.
2. Poco.
3. Regular o creo que en Java la eficiencia no es demasiado importante.
4. Bastante.
5. Mucho.



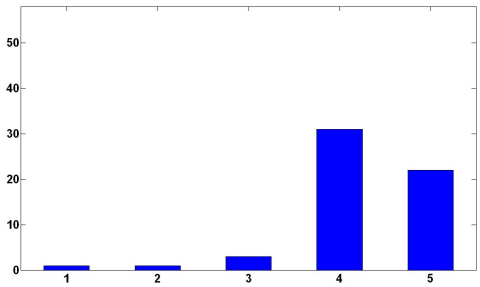
Pregunta 18. ¿Cuánto crees que te ha facilitado jCOLIBRI el desarrollo de la aplicación?

1. Nada.
2. Poco.
3. Regular.
4. Bastante.
5. Mucho.



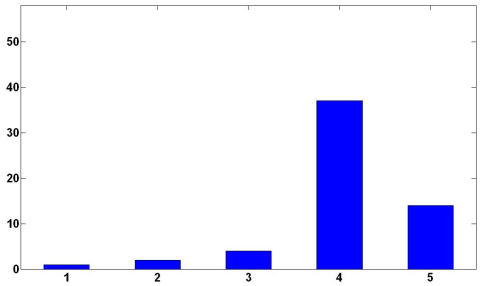
Pregunta 19. Una vez estudiada en clase la teoría sobre los sistemas CBR, ¿crees que jCOLIBRI incluye los métodos necesarios para desarrollar una aplicación CBR (sin contar la GUI)?

1. No, le faltan muchos métodos.
2. Le faltarían algunos métodos.
3. Regular.
4. Incluye casi todos los métodos que necesitaría.
5. Sí, es muy completo.



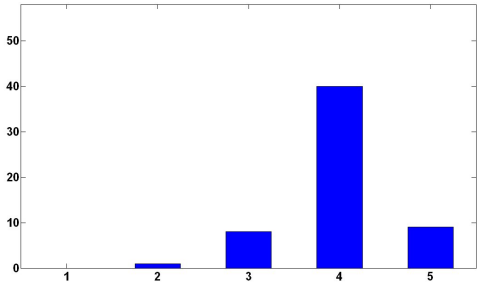
Pregunta 20. ¿Volverías a utilizar jCOLIBRI para desarrollar una aplicación CBR?

1. Nunca.
2. Quizás.
3. No sé.
4. Probablemente.
5. Por supuesto.



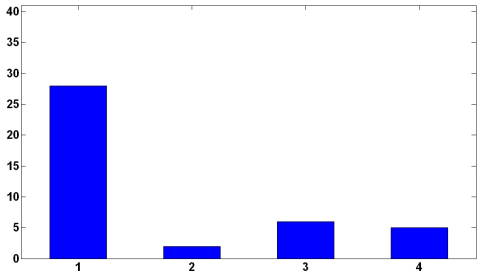
Pregunta 21. ¿Qué nota le pondrías en general?

1. Un cero.
2. Suspenso.
3. Aprobado.
4. Notable.
5. Sobresaliente.



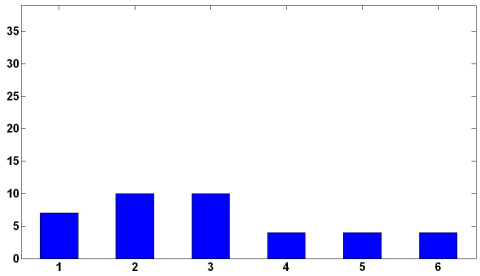
Pregunta 22. ¿Cuál sería su principal virtud? (Respuesta libre)

1. Facilidad para desarrollar aplicaciones CBR.
2. Rapidez de desarrollo para implementar aplicaciones CBR funcionales.
3. Completitud del armazón (funcionalidad).
4. Diseño de clases, arquitectura, tecnologías utilizadas y facilidad de integración.



Pregunta 23. ¿Y su principal inconveniente? (Respuesta libre)

1. Ninguno.
2. Curva de aprendizaje inicial.
3. Complejidad, mucha funcionalidad que dificulta la comprensión.
4. Dificultad en para entender algunas partes del código y depurar.
5. Excesivo tamaño del proyecto.
6. Otros.



A.2 Encuesta Intermedia

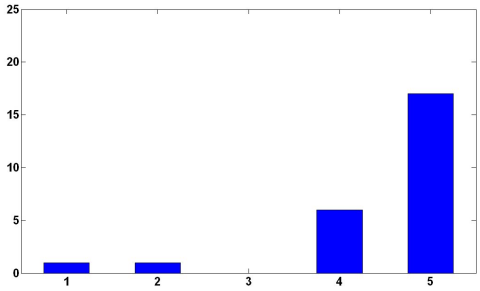
Después de realizar la práctica guiada con jCOLIBRI se explicaron los sistemas CBR de recomendación y se propuso a los alumnos el diseño de un sistema de este tipo. Una vez que los diseños fueron entregados se presentaron los métodos y plantilla incluidos en jCOLIBRI para implementar recomendadores. A continuación, se pidió que rellenaran esta encuesta donde se mide la adecuación de cada una de las plantillas a los diseños planteados por cada alumno.

Esta encuesta se relleno por grupos de trabajo por lo que únicamente se obtuvieron 24 opiniones.

Las diez primeras preguntas corresponden a la adecuación de cada plantilla a su diseño. Los resultados de estas preguntas se encuentran reflejados en la Figura 7.7 de la Sección 7.2.

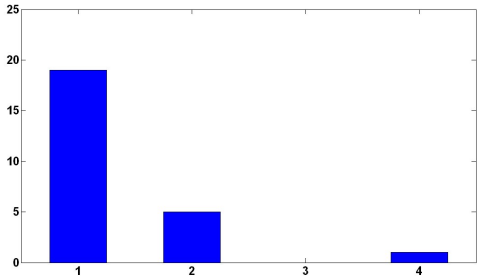
Pregunta 11. Cada plantilla genérica esquematizada en el documento dado se corresponde con uno de los sistemas de ejemplo de jCOLIBRI (ejemplos 17 al 30). Si tu sistema se asemeja a una (o varias) de las plantillas podrías implementarlo reaprovechando uno o varios de los ejemplos de sistemas de jCOLIBRI. ¿Piensas que podrías usar esta forma de implementación (modificando los sistemas ya existentes)?

1. No, utilizaré mi diseño directamente.
2. No inicialmente, pero si encuentro dificultades al implementar mi diseño recurriré a algún recomendador de jCOLIBRI.
3. No lo tengo decidido.
4. Inicialmente sí, pero si encuentro dificultades con el recomendador de jCOLIBRI dejaré de utilizarlo.
5. Sí, utilizaré alguna plantilla de jCOLIBRI.



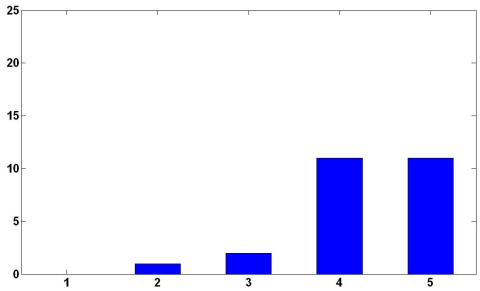
Pregunta 12. ¿Tienes claro cómo empezar a implementar tu recomendador (luego podrías cambiar de idea) o lo decidirás sobre la marcha?

1. Lo tengo claro, modificaré uno de los sistemas de ejemplo para que se ajuste a mi diseño.
2. Lo tengo claro, implementaré con jCOLIBRI pero sin modificar un sistema de ejemplo.
3. Lo tengo claro, implementaré desde cero sin usar jCOLIBRI.
4. No le tengo nada claro, empezaré y decidiré sobre la marcha.



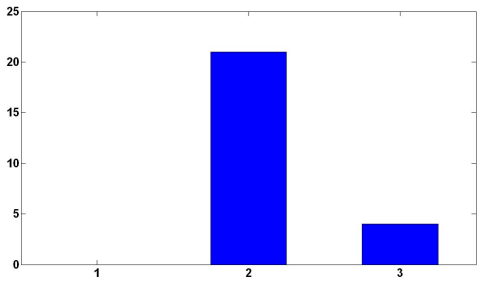
Pregunta 13. ¿Crees que jCOLIBRI incluye la funcionalidad que necesitas para implementar tu diseño? (No tengas en cuenta la interfaz de presentación al usuario)

1. En absoluto.
2. Algunos métodos serían útiles.
3. Quizás, no sé.
4. Podría implementar gran parte del diseño con los métodos de jCOLIBRI.
5. Incluye todo lo que necesito para implementar mi recomendador.



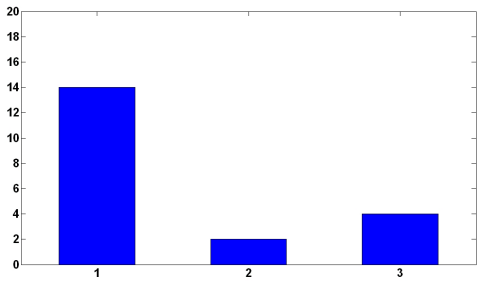
Pregunta 14. Elige la forma en que piensas implementar tu diseño (podrías cambiar de opinión):

1. Implementar en Java sin utilizar jCOLIBRI.
2. Modificar uno o varios de los ejemplos de recomendadores conversacionales dados.
3. Implementar el sistema sin basarme en ningún ejemplo de jCOLIBRI pero sí usando los métodos que ofrece (por ejemplo métodos de cómputo de similitud, conectores, refinamiento de la consulta, elección de la pregunta más discriminante, ...).



Pregunta 15. Entre los requisitos de tu recomendador, ¿Que necesitarías que incluyera jCOLIBRI que no esté ya en el armazón? (Respuesta libre)

1. Nada.
2. Conexión directa al Base de Datos[a].
3. Mayor flexibilidad para crear interfaces gráficas.



[a]Este requisito se añadió al armazón debido a la importante demanda (aunque no se vea reflejado en la encuesta

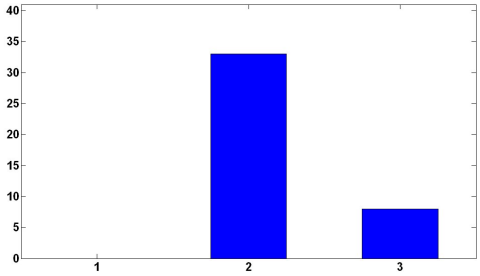
A.3 Encuesta Final

Esta encuesta se realiza una vez implementado el recomendador. En ella se plantea un primer bloque de preguntas para comprobar si la elección del proceso de desarrollo indicada en la encuesta intermedia fue acertada.

En el segundo bloque de preguntas se repiten las preguntas de la encuesta inicial sobre la calidad del diseño y la arquitectura del armazón para poder comparar los resultados. También se repiten las preguntas sobre la documentación y sobre la satisfacción general final. En este caso se rellenaron 40 encuestas por parte de los alumnos.

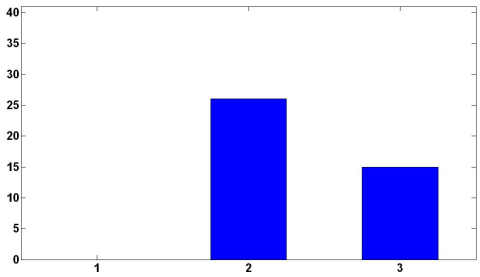
Pregunta 1. ¿Cómo empezaste a desarrollar tu recomendador?

1. Implementando en Java sin utilizar jCOLIBRI.
2. Modificando alguno de los ejemplos de recomendadores de jCOLIBRI
3. Utilizando los métodos de jCOLIBRI (conectores, métodos, medidas de similitud, etc.) pero sin basarme en ningún ejemplo.



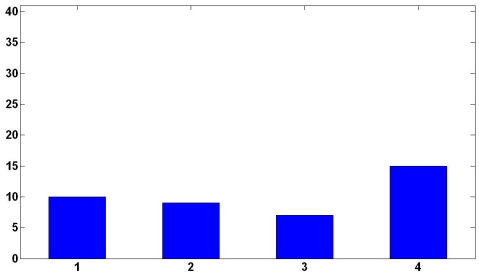
Pregunta 2. ¿Tuviste que cambiar de idea? Es decir, al final ¿cómo acabaste desarrollando el recomendador?

1. Implementando en Java sin utilizar jCOLIBRI.
2. Modificando alguno de los ejemplos de recomendadores de jCOLIBRI.
3. Utilizando los métodos de jCOLIBRI (conectores, métodos, medidas de similitud, etc.) pero sin basarme en ningún ejemplo.



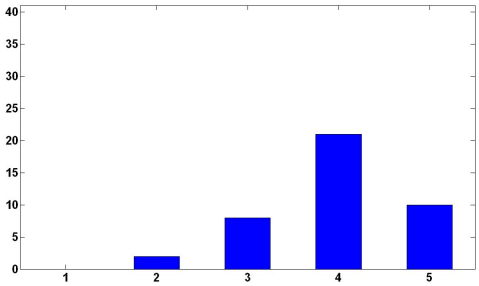
Pregunta 3. ¿Has cambiado el diseño inicial del recomendador sobre la marcha?

1. No.
2. Sí, porque era más fácil utilizar directamente uno de los ejemplos de jCOLIBRI.
3. Sí, porque la implementación era muy compleja y adapté el diseño para utilizar los métodos de jCOLIBRI.
4. Sí, pero por otros motivos.



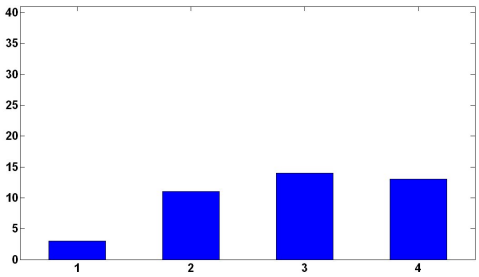
Pregunta 4. ¿Has encontrado en jCOLIBRI los métodos necesarios para realizar el recomendador?

1. No he utilizado jCOLIBRI o sus métodos no me han sido útiles en absoluto.
2. Algunos métodos han resultado útiles.
3. 50 % de los métodos son de jCOLIBRI y el otro 50 % los hemos implementado a mano.
4. La mayor parte de la implementación utiliza los métodos de jCOLIBRI.
5. He reutilizado prácticamente todo.



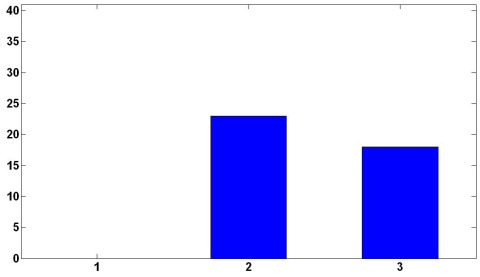
Pregunta 5. ¿Te han sido útiles los ejemplos (plantillas) para implementar tu recomendador? Dicho de otro modo: ¿has reutilizado los ejemplos?

1. No he utilizado ningún ejemplo.
2. He utilizado los ejemplos pero no me han sido tan útiles como pensaba inicialmente.
3. He utilizado los ejemplos y han sido muy útiles.
4. Mi recomendador se basa casi completamente en un ejemplo existente (o en la combinación de varios).



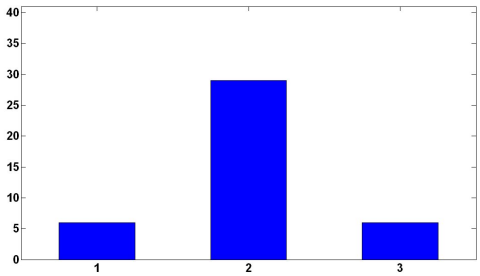
Pregunta 6. Si tuvieras que volver a implementar un recomendador desde el principio, ¿qué opción elegirías?

1. Implementando en Java sin utilizar jCOLIBRI.
2. Modificando alguno de los ejemplos de recomendadores de jCOLIBRI.
3. Utilizando los métodos de jCOLIBRI (conectores, métodos, medidas de similitud, etc.) pero sin basarme en ningún ejemplo.



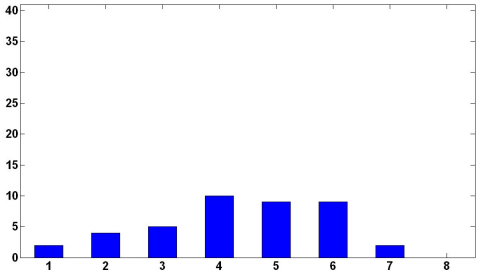
Pregunta 7. En la encuesta anterior te pedimos que compararas las plantillas de jCOLIBRI con tu diseño inicial del recomendador. De esta forma podrías elegir las plantillas más similares para hacer el desarrollo de tu sistema. Una vez que has finalizado el prototipo, ¿qué opinas de la evaluación que hiciste sobre las plantillas?

1. Las plantillas que elegí no se parecían tanto como pensaba al recomendador que quería implementar.
2. La evaluación fue acertada o no he basado mi desarrollo en las plantillas.
3. Las plantillas me han ayudado más de lo que pensaba para hacer el recomendador. Debí evaluarlas más alto.



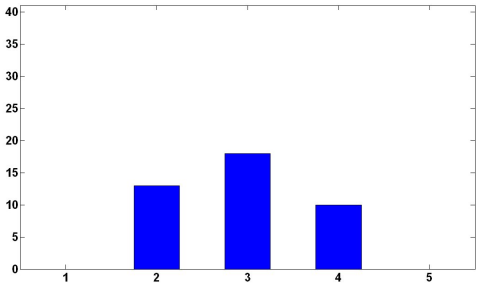
Pregunta 8. ¿Cuál ha sido la parte más difícil del desarrollo?

1. Encontrar los datos o adaptarlos a nuestros requisitos (tanto las bases de datos como ficheros de texto).
2. Configurar el conector para el origen de datos que teníamos.
3. Las dos anteriores.
4. Entender el funcionamiento de los métodos de jCOLIBRI.
5. Configurar los métodos de jCOLIBRI (funciones de similitud, filtrado, etc.).
6. Componer los distintos métodos de jCOLIBRI (p.e. juntar el filtrado con el kNN y con la selección de casos).
7. Nada, todo ha sido fácil.
8. Otra cosa o no he utilizado jCOLIBRI.



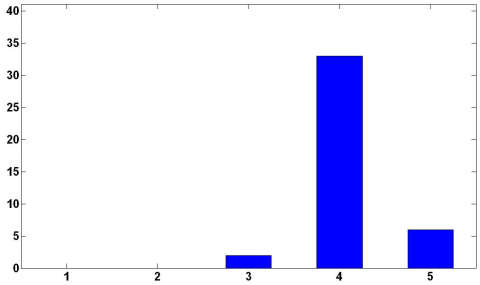
Pregunta 9. ¿Cómo ha sido el proceso de depuración del código según desarrollabas?

1. Ha sido muy difícil.
2. Ha sido algo difícil.
3. Regular.
4. Bastante fácil.
5. Muy fácil.



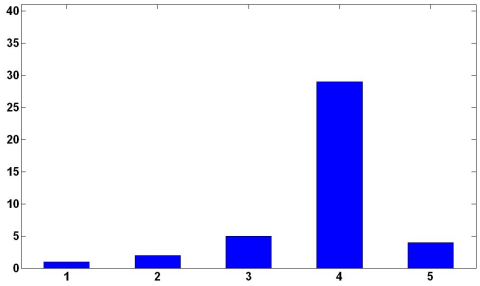
Pregunta 10. ¿Piensas que es acertada la arquitectura de jCOLIBRI donde se separa: la persistencia de los casos (BBDD o txt), su carga en memoria mediante conectores, y su manipulación en memoria por parte de los métodos?

1. No, es una arquitectura muy mala.
2. No creo que sea buena idea utilizar esa separación.
3. No sé o no he utilizado jCOLIBRI como para poder opinar.
4. Creo que es una arquitectura acertada.
5. Es una arquitectura muy buena.



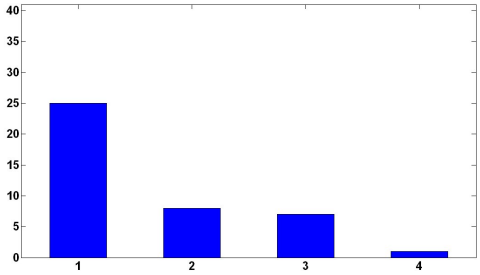
Pregunta 11. ¿Y el diseño de clases?¿Crees que es bueno?

1. No, creo que es muy malo.
2. Tiene bastantes carencias.
3. No sabría qué decir.
4. Creo que es bastante bueno.
5. Creo que es un diseño muy bueno.



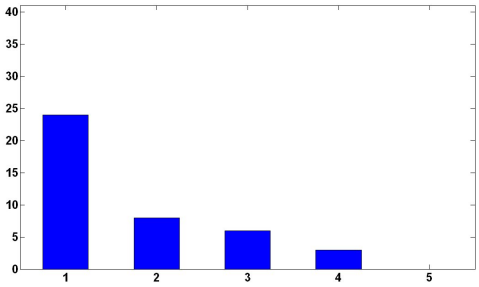
Pregunta 12. ¿Has encontrado fallos/bugs en el código?

1. No.
2. Algún pequeño fallo al que no he dado importancia como para comentárselo a los profesores.
3. Sí, y los han arreglado rápidamente.
4. Sí, y no los han arreglado.



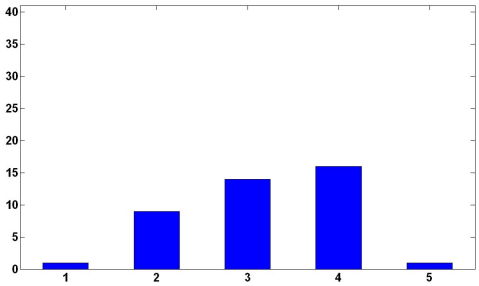
Pregunta 13. ¿Has tenido problemas de eficiencia?

1. No.
2. Sí, debido a la gran cantidad de casos.
3. Sí, debido a la ineficiencia de algunos métodos de jCOLIBRI.
4. Sí, pero es normal en al desarrollar en Java.
5. Sí, pero creo que es culpa del PC donde ejecutaba el recomendador. Era poco potente.



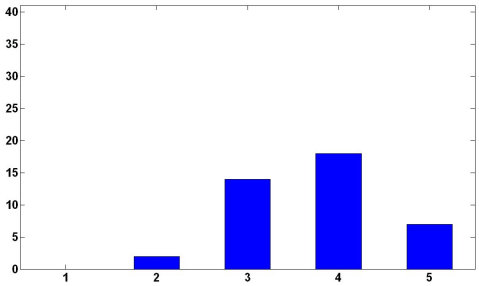
Pregunta 14. ¿Te ha ayudado el tutorial de jCOLIBRI que hay en la página web durante el proceso de desarrollo?

1. Nada.
2. Poco.
3. No lo he leído.
4. Bastante.
5. Mucho.



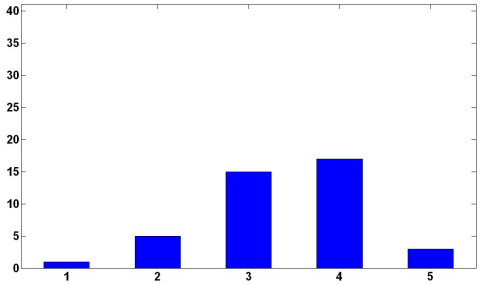
Pregunta 15. ¿Cómo de completa te ha parecido la documentación de jCOLIBRI (tutoriales, ejemplos y API)?

1. Muy escueta.
2. Escueta.
3. Suficiente.
4. Bastante completa.
5. Muy completa.



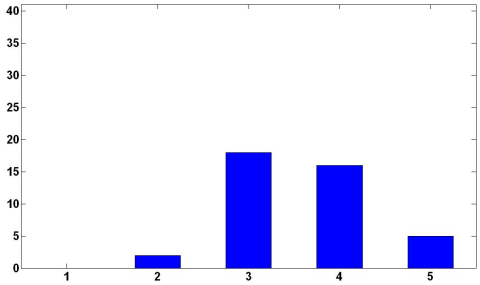
Pregunta 16. ¿Cómo evaluarías la facilidad para encontrar información dentro la documentación de jCOLIBRI (tutoriales, ejemplos y API)?

1. No se encuentra nada.
2. Se encuentra pero cuesta un poco.
3. Normal.
4. Se encuentra bastante bien.
5. Es muy fácil encontrar lo que buscas.



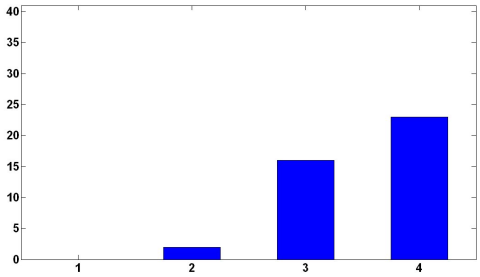
Pregunta 17. ¿Se entienden bien las explicaciones de la documentación?

1. No se entiende nada.
2. Se entiende con dificultad.
3. Normal.
4. Se entiende bastante bien.
5. Está todo muy bien explicado.



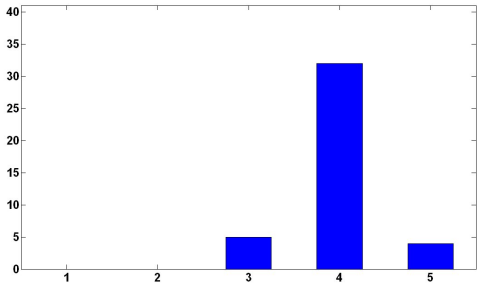
Pregunta 18. ¿Crees que los recomendadores son un área interesante?

1. No.
2. Sí, pero solo desde el punto de vista académico. No les doy demasiada utilidad práctica.
3. Sí, creo que son muy útiles a nivel comercial.
4. Sí, en ambos aspectos (académico y comercial/empresarial).



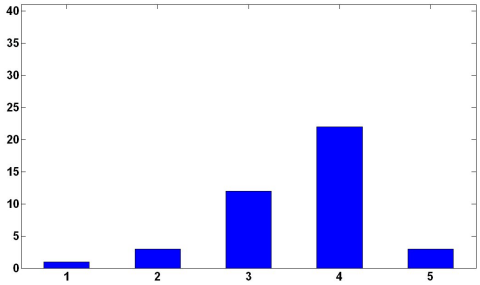
Pregunta 19. ¿Te ha parecido una práctica interesante?

1. En absoluto.
2. Bastante poco.
3. Ni sí ni no.
4. Bastante interesante.
5. Muy interesante.



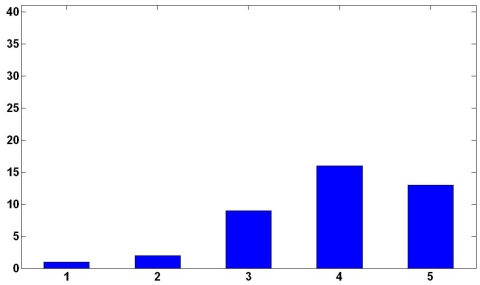
Pregunta 20. En comparación con las prácticas anteriores ¿cómo crees que ha sido el nivel de dificultad?

1. Ha sido bastante más difícil.
2. Un poco más difícil que las anteriores.
3. Igual.
4. Algo más fácil.
5. Mucho más fácil.



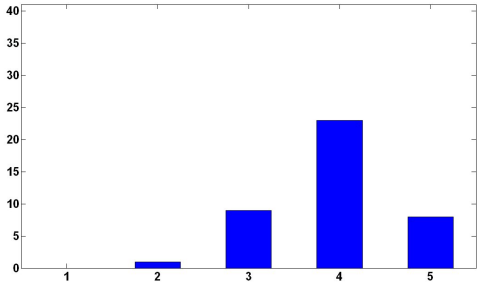
Pregunta 21. Cuando has tenido problemas con el código, ¿cómo ha sido el nivel de asistencia técnica por parte de los profesores?

1. Bastante malo y he tenido que buscarme la vida.
2. Podría ser mejor.
3. Creo que han sido correctos en todo momento.
4. Me han ayudado bastante.
5. Me han solucionado todos los problemas muy eficientemente.



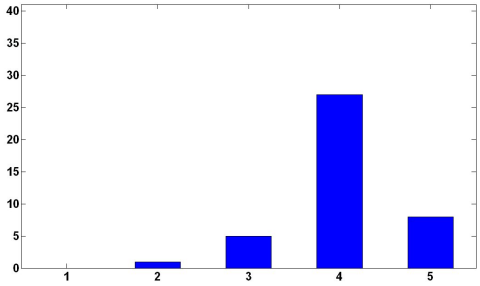
Pregunta 22. ¿Volverías a utilizar jCOLIBRI para desarrollar una aplicación CBR (esto incluye recomendadores)?

1. Nunca.
2. Quizás.
3. No sé.
4. Probablemente.
5. Por supuesto.



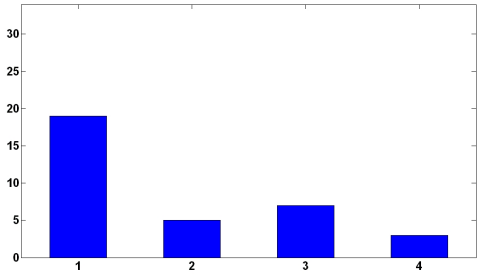
Pregunta 23. ¿Qué nota final le pondrías a jCOLIBRI?

1. Un cero.
2. Suspenso.
3. Aprobado.
4. Notable.
5. Sobresaliente.



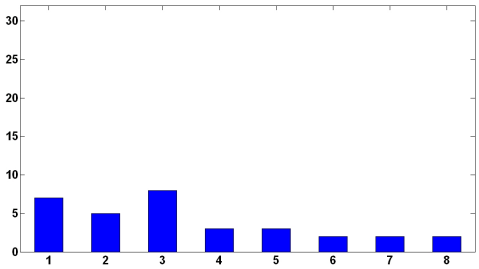
Pregunta 24. ¿Qué es lo que más te ha gustado?

1. Facilidad de desarrollo gracias a la gran variedad de ejemplos/plantillas.
2. Variedad de métodos disponibles.
3. Diseño y arquitectura.
4. Facilidad de uso en general.



Pregunta 25. ¿Y lo que menos?

1. Nada.
2. Curva de aprendizaje.
3. Comprensión del diseño de clases, métodos y configuración.
4. Dificultad de depuración.
5. Carencias en la documentación.
6. Excesivo tamaño del proyecto.
7. Poca flexibilidad en las interfaces gráficas.
8. Deficiencias de diseño o métodos.



Apéndice B

CBROnto

Este anexo incluye el pseudocódigo de la ontología CBROnto. Este código está dividido en dos partes distintas: el dedicado a la representación de las plantillas y el de representación de los métodos. Ambas partes de la ontología se presentan por separado para mejorar su comprensión. Estas dos partes se enlazan a través de la propiedad *solves* que indica qué tarea es resuelta por un determinado método.

Además de utilizar pseudocódigo, se ha eliminado cierta información no crucial como las clases o instancias disjuntas.

B.1 Representación de las plantillas

El siguiente listado muestra el código OWL que permite la representación de las plantillas. En la parte final del mismo se muestran las instancias específicas para representar plantillas de sistemas de recomendación.

Listado B.1: Código para la representación de las plantillas

```
/////////////////////////////////////////////
// CONCEPTOS
/////////////////////////////////////////////

Class: CBRApplicationTemplate
    SubClassOf:
        owl:Thing,
        templateAuthor min 1 Literal,
        templateSubType max 1 Literal,
        hasPreCycle exactly 1 owl:Thing,
        hasCycle exactly 1 owl:Thing,
        hasPostCycle exactly 1 owl:Thing,
        templateType exactly 1 Literal

Class: FinalCBRApplicationTemplate
    SubClassOf:
        CBRApplicationTemplate,
        expands exactly 1 owl:Thing

Class: GenericCBRApplicationTemplate
    SubClassOf:
        CBRApplicationTemplate

Class: TemplateStage
    SubClassOf:
        owl:Thing,
        hasThumb max 1 Literal,
        hasFlow exactly 1 owl:Thing

Class: PreCycle
    SubClassOf: TemplateStage

Class: Cycle
    SubClassOf: TemplateStage

Class: PostCycle
    SubClassOf: TemplateStage

Class: Tool
    SubClassOf: TemplateStage

Class: ControlConstruct
```

```
Class: Sequence
    SubClassOf:
        ControlConstruct,
        components exactly 1 owl:Thing

Class: ControlConstructList
    SubClassOf:
        owl:Thing,
        first exactly 1 owl:Thing

Class: Task
    SubClassOf: ControlConstruct

Class: GenericTask
    SubClassOf: ControlConstruct

Class: ConditionalTask
    SubClassOf: Task

Class: If-Then-Else
    SubClassOf:
        ControlConstruct,
        ifCondition exactly 1 owl:Thing

Class: Begin
    SubClassOf: ControlConstruct

Class: Finish
    SubClassOf: ControlConstruct

Class: Method

//////////////////////////////////////////////
// PROPIEDADES
//////////////////////////////////////////////

DataProperty: templateSubType
    Domain: CBRApplicationTemplate
    Range:  string

DataProperty: templateType
    Domain: CBRApplicationTemplate
    Range:  string

DataProperty: templateAuthor
    Domain: CBRApplicationTemplate
    Range:  string
```

```
DataProperty: templateDescription
    Domain: CBRApplicationTemplate
    Range:  string

ObjectProperty: expands
    Domain: FinalCBRApplicationTemplate
    Range:  GenericCBRApplicationTemplate

DataProperty: hasThumb
    Domain: TemplateStage
    Range:  string

ObjectProperty: hasPreCycle
    Domain: CBRApplicationTemplate
    Range:  PreCycle

ObjectProperty: hasCycle
    Domain: CBRApplicationTemplate
    Range:  Cycle

ObjectProperty: hasPostCycle
    Domain: CBRApplicationTemplate
    Range:  PostCycle

ObjectProperty: hasTool
    Domain: CBRApplicationTemplate
    Range:  Tool

ObjectProperty: hasFlow
    Domain: TemplateStage
    Range:  Sequence

ObjectProperty: components
    Domain: Sequence
    Range:  ControlConstructList

ObjectProperty: first
    Domain: ControlConstructList
    Range:  ControlConstruct

ObjectProperty: rest
    Domain: ControlConstructList
    Range:  ControlConstructList

ObjectProperty: ifCondition
    Domain: If-Then-Else
    Range:  ConditionalTask

ObjectProperty: then
```

```
    Domain: If-Then-Else
    Range:  ControlConstruct

ObjectProperty: else
    Domain: If-Then-Else
    Range:  ControlConstruct

ObjectProperty: solves
    Domain: Method
    Range:  Task

//////////////////////////////////////////////
// INSTANCIAS
//////////////////////////////////////////////

Individual: nil
     Types: ControlConstructList

//INSTANCIAS ESPECÍFCIAS RECOMENDADORES

Individual: DisplayCases
Individual: SelectQuestion
Individual: CreateComplexQuery
Individual: Scoring
Individual: SelectCases
Individual: UpdateTabu
Individual: Filtering
Individual: FormFilling
Individual: ReadProfile
Individual: RemoveTabu
Individual: AskQuestion
     Types: Task

Individual: Retrieval
Individual: Display
Individual: Iterated_Preference_Elicitation
Individual: One-Off_Preference_Elicitation
     Types: GenericTask


Individual: FinishUserQuits
Individual: FinishUserBuys
Individual: End
     Types: Finish

Individual: Continue
Individual: EvalDisplayCases
Individual: BuyOrQuit
     Types: ConditionalTask
```

B.2 Ejemplo de representación de una plantilla

El siguiente listado muestra un ejemplo de representación de una plantilla a través de CBROnto. Este código se corresponde con la plantilla mostrada en la parte inferior de la Figura 6.9 (pág. 208), que también es visualizado en la Figura 6.10 (pág. 210).

Como puede observarse en el código, las listas de tareas se representan de forma recursiva como un elemento de cabeza (`first`) seguido de una cola (`rest`) que a su vez tendrá cabeza y cola hasta llegar al elemento `nil`. En este caso el preciclo y postciclo de la plantilla están vacíos y no contienen ninguna tarea.

Listado B.2: Ejemplo de representación de una plantilla

```
Individual: Template2
     Types: FinalCBRApplicationTemplate
     Facts:
        hasPreCycle  Template2_PreCycle,
        hasPostCycle  Template2_PostCycle,
        hasCycle  Template2_Cycle
        templateSubType  "A",
        templateType  "Conversatonal",
        templateAuthor  "Derek Bridge",
        templateDescription  "Conversational (type A) flats
            recommender using form-filling and NN retrieval"


//////////////////////////////////////////////
// PRECYCLE
//////////////////////////////////////////////

Individual: Template2_PreCycle
     Types: PreCycle
     Facts:
        hasFlow  Template2_PreCycle_Sequence_1
        hasThumb  "templates\Template2_PreCycle.jpg"

Individual: Template2_PreCycle_Sequence_1
     Types: Sequence
     Facts:
        components  Template2_PreCycle_Sequence_1List_1

Individual: Template2_PreCycle_Sequence_1List_1
     Types: ControlConstructList
     Facts:
        rest  Template2_PreCycle_Sequence_1List_2,
        first  Template2_PreCycle_Begin

Individual: Template2_PreCycle_Begin
     Types: Begin
```

```
Individual: Template2_PreCycle_Sequence_1List_2
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  End

//////////////////////////////////////////////
// CYCLE
//////////////////////////////////////////////

Individual: Template2_Cycle
     Types: Cycle
     Facts:
        hasFlow  Template2_Cycle_Sequence_2
        hasThumb  "templates\Template2_Cycle.jpg"

Individual: Template2_Cycle_Sequence_2
     Types: Sequence
     Facts:
        components  Template2_Cycle_Sequence_2List_1

Individual: Template2_Cycle_Sequence_2List_1
     Types: ControlConstructList
     Facts:
        rest  Template2_Cycle_Sequence_2List_2,
        first  Template2_Cycle_Begin

//BEGIN
Individual: Template2_Cycle_Begin
     Types: Begin

//FORMFILLING
Individual: Template2_Cycle_Sequence_2List_2
     Types: ControlConstructList
     Facts:
        rest  Template2_Cycle_Sequence_2List_3,
        first  FormFilling


Individual: Template2_Cycle_Sequence_2List_3
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  Template2_Cycle_Sequence_3
//DO
Individual: Template2_Cycle_Sequence_3
     Types: Sequence
     Facts:
        components  Template2_Cycle_Sequence_3List_1
```

```
//SCORING
Individual: Template2_Cycle_Sequence_3List_1
     Types: ControlConstructList
     Facts:
        rest  Template2_Cycle_Sequence_3List_2,
        first  Scoring

//SELECTCASES
Individual: Template2_Cycle_Sequence_3List_2
     Types: ControlConstructList
     Facts:
        rest  Template2_Cycle_Sequence_3List_3,
        first  SelectCases

//DISPLAYCASES
Individual: Template2_Cycle_Sequence_3List_3
     Types: ControlConstructList
     Facts:
        rest  Template2_Cycle_Sequence_3List_4,
        first  DisplayCases

Individual: Template2_Cycle_Sequence_3List_4
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  Template2_Cycle_If_1

//IF (Continue)
Individual: Template2_Cycle_If_1
     Types: If-Then-Else
     Facts:
        then  Template2_Cycle_Sequence_4,
        ifCondition  Continue,
        else  Template2_Cycle_Sequence_5

//THEN
Individual: Template2_Cycle_Sequence_4
     Types: Sequence
     Facts:
        components  Template2_Cycle_Sequence_4List_1

Individual: Template2_Cycle_Sequence_4List_1
     Types: ControlConstructList
     Facts:
        rest  Template2_Cycle_Sequence_4List_2,
        first  FormFilling

//GO -> DO
```

```
Individual: Template2_Cycle_Sequence_4List_2
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  Template2_Cycle_Sequence_3

//ELSE
Individual: Template2_Cycle_Sequence_5
     Types: Sequence
     Facts:
        components  Template2_Cycle_Sequence_5List_1

Individual: Template2_Cycle_Sequence_5List_1
     Types: ControlConstructList
     Facts:
        rest  :nil,
        first  Template2_Cycle_If_2

//IF (BuyOrQuit)
Individual: Template2_Cycle_If_2
     Types: If-Then-Else
     Facts:
        then  Template2_Cycle_Sequence_6,
        ifCondition  BuyOrQuit,
        else  Template2_Cycle_Sequence_7

//THEN
Individual: Template2_Cycle_Sequence_6
     Types: Sequence
     Facts:
        components  Template2_Cycle_Sequence_6List_1

Individual: Template2_Cycle_Sequence_6List_1
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  FinishUserBuys

//ELSE
Individual: Template2_Cycle_Sequence_7
     Types: Sequence
     Facts:
        components  Template2_Cycle_Sequence_7List_1

Individual: Template2_Cycle_Sequence_7List_1
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  FinishUserQuits
```

```
////////////////////////////////////////////////
// POSTCYCLE
////////////////////////////////////////////////

Individual: Template2_PostCycle
     Types: PostCycle
     Facts:
        hasFlow  Template2_PostCycle_Sequence_8
        hasThumb  "templates\Template2_PostCycle.jpg"

Individual: Template2_PostCycle_Sequence_8
     Types: Sequence
     Facts:
        components  Template2_PostCycle_Sequence_8List_1

Individual: Template2_PostCycle_Sequence_8List_1
     Types: ControlConstructList
     Facts:
        rest  Template2_PostCycle_Sequence_8List_2,
        first  Template2_PostCycle_Begin

Individual: Template2_PostCycle_Begin
     Types: Begin

Individual: Template2_PostCycle_Sequence_8List_2
     Types: ControlConstructList
     Facts:
        rest  nil,
        first  End
```

B.3 Representación de los métodos

En el siguiente listado se muestra la parte de CBROnto encargada de representar los métodos. Este código es el que se representa en la Figura 6.11.

Listado B.3: Código para la representación de los métodos

```
//////////////////////////////////////////////
// CONCEPTOS
//////////////////////////////////////////////

Class: Method

Class: MethodRequirements

Class: PreCondition
    SubClassOf: MethodRequirements

Class: PostCondition
    SubClassOf: MethodRequirements

Class: MethodSignature

Class: Input
    SubClassOf: MethodSignature

Class: Output
    SubClassOf: MethodSignature

Class: Parameter

Class: ParameterNature

Class: Getter

Class: MethodType

//////////////////////////////////////////////
// PROPIEDADES
//////////////////////////////////////////////

ObjectProperty: solves
    Domain: Method
    Range:  Task

ObjectProperty: methodType
    Domain: Method
    Range:  MethodType

DataProperty: methodDescription
```

```
    Domain: Method
    Range:  string

DataProperty: methodRealPath
    Domain: Method
    Range:  string

ObjectProperty: hasPreCondition
    Domain: Method
    Range:  PreCondition

ObjectProperty: hasPostCondition
    Domain: Method
    Range:  PostCondition

ObjectProperty: hasSignature
    Domain: MethodRequirements
    Range:  MethodSignature

ObjectProperty: hasParameter
    Domain: MethodSignature
    Range:  Parameter

DataProperty: parameterName
    Domain: Parameter
    Range:  string

DataProperty: parameterDescription
    Domain: Parameter
    Range:  string

ObjectProperty: parameterNature
    Domain: Parameter
    Range:  ParameterNature

ObjectProperty: parameterType
    Domain: Parameter
    Range:  Type

ObjectProperty: parameterPreferredType
    Domain: Parameter
    Range:  Type

ObjectProperty: hasGetMethod
    Domain: Type
    Range:  Getter

ObjectProperty: gets
    Domain: Getter
```

```
    Range:  Type

DataProperty: getterDescription
    Domain: Getter
    Range:  string

//////////////////////////////////////////////
// INSTANCIAS
//////////////////////////////////////////////

Individual: INPUT_USER
Individual: OUTPUT_RETURNED
Individual: INPUT_BOTH
Individual: INPUT_METHOD
Individual: OUTPUT_REFERENCED
    Types:  ParameterNature

Individual: NORMAL
Individual: CONDITIONAL
     Types: MethodType
```

B.4 Ejemplo de representación de un método

A continuación se incluye el fragmento de código con representación real mediante CBROnto de uno de los métodos del armazón. Esta representación se obtiene automáticamente de las anotaciones en el código fuente.

En este caso se ha elegido la representación del método `GreedySelection` que muestra en la Figura 6.13 (pág. 223) a partir de la anotación del Listado 6.3 (pág. 221).

Listado B.4: Ejemplo de representación de un método

```
Individual: GreedySelection
     Types: Method
     Facts:
        solves  SelectCasesmethod
        hasPreCondition  GreedySelection_Precondition,
        methodType  NORMAL,
        hasPostCondition  GreedySelection_PostCondition,
        methodRealPath  "jcolibri2.source2onto.example.
            SelectCasesMethods.greedySelection",
        Description  "This method incrementally builds a retrieval set
            , R. During each step the remaining cases are ordered
            according to ..."

//PRECONDITION

Individual: GreedySelection_Precondition
     Types: PreCondition
     Facts:
        hasSignature  GreedySelection_Input

Individual: GreedySelection_Input
     Types: Input
     Facts:
        hasParameter  GreedySelection_Input_simConfig,
        hasParameter  GreedySelection_Input_query,
        hasParameter  GreedySelection_Input_k,
        hasParameter  GreedySelection_Input_cases

Individual: GreedySelection_Input_simConfig
     Types: Parameter
     Facts:
        parameterPreferredType  iNNConfig,
        parameterType  iNNConfig
        parameterDescription  "NN configuration object",
        parameterName  "simConfig"

Individual: GreedySelection_Input_query
     Types: Parameter
     Facts:
```

```
        parameterPreferredType  iCBRQuery,
        parameterType  iCBRQuery
        parameterDescription  "Query used to compute diversity",
        parameterName  "query"

Individual: GreedySelection_Input_k
     Types: Parameter
     Facts:
        parameterPreferredType  iInteger,
        parameterType  iInteger
        parameterDescription  "Number of cases to select",
        parameterName  "k"

Individual: GreedySelection_Input_cases
     Types: Parameter
     Facts:
        parameterPreferredType  iSmallRetrievalResultCollection,
        parameterType  iRetrievalResultCollection
        parameterDescription  "Cases list where select",
        parameterName  "cases"

//POSTCONDITION

Individual: GreedySelection_PostCondition
     Types: PostCondition
     Facts:
        hasSignature  GreedySelection_Output

Individual: GreedySelection_Output
     Types: Output
     Facts:
        hasParameter  GreedySelection_Output_RETURNED_VALUE

Individual: GreedySelection_Output_RETURNED_VALUE
     Types: Parameter
     Facts:
        parameterType  iCBRCaseCollection
        parameterDescription  "Selected cases"
```

B.5 Jerarquía de tipos

Finalmente, el siguiente listado muestra la jerarquía de tipos básica de CBROnto.

Listado B.5: Jerarquía de tipos básica

```
//////////////////////////////////////////////
// CONCEPTOS
//////////////////////////////////////////////

Class: Type

Class: String
    SubClassOf: Type

Class: CBRCaseBase
    SubClassOf: Type

Class: PearsonMatrixCaseBase
    SubClassOf: CBRCaseBase

Class: Void
    SubClassOf: Type

Class: Collection
    SubClassOf: Type

Class: RetrievalResultCollection
    SubClassOf: Collection

Class: LargeRetrievalResultCollection
    SubClassOf: RetrievalResultCollection

Class: MediumRetrievalResultCollection
    SubClassOf: LargeRetrievalResultCollection

Class: SmallRetrievalResultCollection
    SubClassOf: MediumRetrievalResultCollection

Class: AttributeCollection
    SubClassOf: Collection

Class: CBRCaseCollection
    SubClassOf: Collection

Class: CDRSet
    SubClassOf: CBRCaseCollection

Class: LargeCBRCaseCollection
    SubClassOf: CBRCaseCollection
```

```
Class: MediumCBRCaseCollection
    SubClassOf: LargeCBRCaseCollection

Class: SmallCBRCaseCollection
    SubClassOf: MediumCBRCaseCollection

Class: CritiqueOptionCollection
    SubClassOf: Collection

Class: Integer
    SubClassOf: Type

Class: Attribute
    SubClassOf: Type

Class: RetrievalResult
    SubClassOf: Type

Class: CBRQuery
    SubClassOf: Type

Class: CBRCase
    SubClassOf: CBRQuery

Class: Double
    SubClassOf: Type

Class: Connector
    SubClassOf: Type

Class: FilterConfig
    SubClassOf: Type

Class: Map
    SubClassOf: Type

Class: Attribute2DoubleMap
    SubClassOf: Map

Class: Attribute2StringMap
    SubClassOf: Map

Class: CritiqueOption
    SubClassOf: Type

Class: NNConfig
    SubClassOf: Type
```

```
Class: Boolean
    SubClassOf: Type

Class: UserChoice
    SubClassOf: Type

Class: CriticalUserChoice
    SubClassOf: UserChoice

//////////////////////////////////////////////
// INSTANCIAS
//////////////////////////////////////////////

Individual: iCDRSet
     Types: CDRSet

Individual: iBoolean
     Types: Boolean

Individual: iRetrievalResult
     Types: RetrievalResult

Individual: iAttribute2DoubleMap
     Types: Attribute2DoubleMap

Individual: iSmallRetrievalResultCollection
     Types: SmallRetrievalResultCollection

Individual: iSmallCBRCaseCollection
     Types: SmallCBRCaseCollection

Individual: iCriticalUserChoice
     Types: CriticalUserChoice

Individual: iCBRCaseCollection
     Types: CBRCaseCollection

Individual: iLargeRetrievalResultCollection
     Types: LargeRetrievalResultCollection

Individual: iCBRQuery
     Types: CBRQuery

Individual: iInteger
     Types: Integer

Individual: iMediumCBRCaseCollection
     Types: MediumCBRCaseCollection
```

```
Individual: iCritiqueOptionCollection
     Types: CritiqueOptionCollection

Individual: iLargeCBRCaseCollection
     Types: LargeCBRCaseCollection

Individual: iString
     Types: String

Individual: iCritiqueOption
     Types: CritiqueOption

Individual: iMediumRetrievalResultCollection
     Types: MediumRetrievalResultCollection

Individual: iAttributeCollection
     Types: AttributeCollection

Individual: iDouble
     Types: Double

Individual: iCBRCaseBase
     Types: CBRCaseBase

Individual: iUserChoice
     Types: UserChoice

Individual: iAttribute
     Types: Attribute

Individual: iCBRCase
     Types: CBRCase

Individual: iFilterConfig
     Types: FilterConfig

Individual: iRetrievalResultCollection
     Types: RetrievalResultCollection

Individual: iNNConfig
     Types: NNConfig

Individual: iPearsonMatrixCaseBase
     Types: PearsonMatrixCaseBase

Individual: iConnector
     Types: Connector

Individual: iAttribute2StringMap
```

```
     Types: Attribute2StringMap

Individual: iVoid
     Types: Void
```

Bibliografía

A. Aamodt, 1990. *Knowledge intensive case-based reasoning and sustained learning.* En *Proceedings of the ninth European Conference on Artificial Intelligence – (ECAI-90)*, páginas 1–6.

A. Aamodt, 2001. *Modeling the knowledge contents of cbr systems.* En Weber y Gresse von Wangenheim (2001), páginas 32–37.

A. Aamodt, 2004. *Knowledge-intensive case-based reasoning in creek.* En Funk y González-Calero (2004), páginas 1–15.

A. Aamodt y E. Plaza, 1994. *Case-based reasoning: Foundational issues, methodological variations, and system approaches.* AI Communications, 7(i).

C. Abásolo, E. Plaza, y J.-L. Arcos, 2002. *Components for Case-Based Reasoning Systems.* Lecture Notes in Computer Science, 2504. ISSN 0302-9743.

T. L. Acorn y S. H. Walden, 1992. *Smart: Support: Management automated reasoning technology for compaq customer service.* En A. C. Scott y P. Klahr, editores, *IAAI*, páginas 3–18. AAAI. ISBN 0-262-69155-8.

D. W. Aha, editor, 2002. *6th European Conference on Case Based Reasoning, ECCBR 2002, Aberdeen, Scotland, UK, September 4-7, 2002, Workshop Proceedings.*

D. W. Aha, L. Breslow, y H. M. noz Avila, 2001. *Conversational case-based reasoning.* Applied Intelligence, 14(1):9–32.

D. W. Aha, D. MsSherry, y Q. Yang, 2006. *Advances in conversational case-based reasoning.* The Knowledge Engineering Review, 20(03):247–254.

D. W. Aha y I. Watson, editores, 2001. *Case-Based Reasoning Research and Development, 4th International Conference on Case-Based Reasoning, ICCBR 2001, Vancouver, BC, Canada, July 30 - August 2, 2001, Proceedings*, tomo 2080 de *Lecture Notes in Computer Science.* Springer. ISBN 3-540-42358-3.

V. Aleven, 1997. *Teaching Case-Based Argumentation Through a Model and Examples.* Tesis Doctoral, University of Pittsburgh.

K.-D. Althoff, E. Auriol, R. Barletta, y M. Manago, 1995. *A Review of Industrial Case-Based Reasoning Tools.* AI Intelligence, Oxford. ISBN 1-898804-01-X.

K.-D. Althoff, R. Bergmann, M. Minor, y A. Hanft, editores, 2008. *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings*, tomo 5239 de *Lecture Notes in Computer Science*. Springer. ISBN 978-3-540-85501-9. ISSN 0302-9743.

D. Alur, D. Malks, y J. Crupi, 2001. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall PTR, Upper Saddle River, NJ, USA. ISBN 0130648841.

L. Aroyo, P. D. Bra, y V. I. Chepegin, 2004. *Semantic web-based adaptive hypermedia*. En C. Bussler, S. Decker, D. Schwabe, y O. Pastor, editores, *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, tomo 105 de *CEUR Workshop Proceedings*.

K. D. Ashley, 1991. *Reasoning with cases and hypotheticals in hypo*. International Journal of Man-Machine Studies, 34(6):753–796.

K. D. Ashley y V. Aleven, 1993. *A logical representation for relevance criteria*. En S. Wess, K.-D. Althoff, y M. M. Richter, editores, *EWCBR*, tomo 837 de *Lecture Notes in Computer Science*, páginas 338–352. Springer. ISBN 3-540-58330-0.

K. D. Ashley y D. G. Bridge, editores, 2003. *Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR 2003, Trondheim, Norway, June 23-26, 2003, Proceedings*, tomo 2689 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-40433-3.

F. Baader, D. Calvanese, D. L. McGuinness, D.Ñardi, y P. F. Patel-Schneider, editores, 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press. ISBN 0-521-78176-0.

C. Bauer y G. King, 2006. *Java Persistence with Hibernate*. Manning Publications Co., Greenwich, CT, USA. ISBN 1932394885.

V. R. Benjamins y D. Fensel, 1998. *Editorial: problem-solving methods*. International Journal of Human-Computer Studies, 49(4):305–313.

R. Bergmann, 2002. *Experience Management: Foundations, Development Methodology, and Internet-Based Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 3540441913.

R. Bergmann, M. M. Richter, S. Schmitt, A. Stahl, y I. Vollrath, 2001. *Utility-oriented matching: A new research direction for case-based reasoning*. En H.-P. Schnurr, S. Staab, R. Studer, G. Stumme, y Y. Sure, editores, *Professionelles Wissensmanagement – Erfahrungen und Visionen (Includes Proceedings of the 9th German Workshop on Case-Based Reasoning, GWCBR 2001), Baden-Baden, Germany*, páginas 264–274. Shaker-Verlag, Aachen.

R. Bergmann y A. Stahl, 1998. *Similarity measures for object-oriented case representations*. En Smyth y Cunningham (1998), páginas 25–36.

S. R. Bhatta y A. K. Goel, 1997. *A functional theory of design patterns*. En *Proceedings of the International Joint Conference on Artificial Intelligence*, páginas 294–300.

E. Blanzieri y L. Portinale, editores, 2000. *Advances in Case-Based Reasoning, 5th European Workshop, EWCBR 2000, Trento, Italy, September 6-9, 2000, Proceedings*, tomo 1898 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-67933-2.

S. Bogaerts y D. Leake, 2005. *Iucbrf: A framework for rapid and modular case-based reasoning system development.* Informe Técnico 617, Indiana University, http://www.cs.indiana.edu/ sbogaert/CBR/IUCBRF.pdf.

K. Branting, J. C. Lester, y B. W. Mott, 2004. *Dialogue Management for Conversational Case-Based Reasoning.* En Funk y González-Calero (2004), páginas 77–90.

D. Bridge, M. H. Göker, L. McGinty, y B. Smyth, 2006. *Case-based recommender systems.* Knowledge Engineering Review, 20(3):315–320. ISSN 0269-8889.

D. G. Bridge, 1998. *Defining and combining symmetric and asymmetric similarity measures.* En Smyth y Cunningham (1998), páginas 52–63.

D. G. Bridge, 2002. *Towards conversational recommender systems: A dialogue grammar approach.* En Aha (2002), páginas 9–22.

H. Brighton y C. Mellish, 2002. *Advances in instance selection for instance-based learning algorithms.* Data Mining and Knowledge Discovery, 6(2):153–172. ISSN 1384-5810.

M. Brown, C. Förtsch, y D. Wissmann, 1998. *Feature extraction - the bridge from case-based reasoning to information retrieval.* Proceedings of the 6th German Workshop on Case-Based Reasoning (GWCBR'98).

S. Brüninghaus y K. D. Ashley, 1999. *Bootstrapping case base development with annotated case summaries.* En K.-D. Althoff, R. Bergmann, y K. Branting, editores, *ICCBR*, tomo 1650 de *Lecture Notes in Computer Science*. Springer. ISBN 3-540-66237-5.

S. Brüninghaus y K. D. Ashley, 2001. *The role of information extraction for textual cbr.* En Aha y Watson (2001), páginas 74–89.

S. Brüninghaus y K. D. Ashley, 2006. *Progress in Textual Case-Based Reasoning: Predicting the Outcome of Legal Cases from Text.* En *AAAI*. AAAI Press.

R. Burke, 2002. *Interactive critiquing forcatalog navigation in e-commerce.* Knowledge Engineering Review, 18(3-4):245–267. ISSN 0269-2821.

R. D. Burke, K. J. Hammond, V. A. Kulyukin, S. L. Lytinen, N. Tomuro, y S. Schoenberg, 1997a. *Question answering from frequently asked question files: Experiences with the FAQ finder system.* AI Magazine, 18(2)(TR-97-05):57–66.

R. D. Burke, K. J. Hammond, y B. C. Young, 1997b. *The FindMe approach to assisted browsing.* IEEE Expert, 12(5):32–40.

P. Castells, M. Fernández, y D. Vallet, 2007. *An adaptation of the vector-space model for ontology-based information retrieval.* IEEE Transactions on Knowledge and Data Engineering, 19(2):261–272. ISSN 1041-4347.

B. Chandrasekaran, J. R. Josephson, y V. R. Benjamins, 1999. *Ontologies: What are they? wy do we need them?* IEEE Intelligent Systems and their applications, 14(1):20–26.

D. Connolly, F. v. Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, y L. A. Stein, 2001. *DAML+OIL Reference Description.* World Wide Web Consortium.

M. T. Cox, H. Muñoz-Avila, y R. Bergmann, 2006. *Case-based planning.* The Knowledge Engineering Review, 20(03):283–287.

S. Craw y A. D. Preece, editores, 2002. *Advances in Case-Based Reasoning, 6th European Conference, ECCBR 2002 Aberdeen, Scotland, UK, September 4-7, 2002, Proceedings*, tomo 2416 de *Lecture Notes in Computer Science.* Springer. ISBN 3-540-44109-3.

M. Crubézy y M. A. Musen, 2004. *Ontologies in support of problem solving.* En S. Staab y R. Studer, editores, *Handbook on Ontologies*, International Handbooks on Information Systems, páginas 321–342. Springer. ISBN 3-540-40834-7.

H. Cunningham, D. Maynard, K. Bontcheva, y V. Tablan, 2002. *GATE: A framework and graphical development environment for robust NLP tools and applications.* En *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics.*

P. Cunningham, D. Doyle, y J. Loughrey, 2003. *An evaluation of the usefulness of case-based explanation.* En Ashley y Bridge (2003), páginas 122–130.

J. J. Daniels y E. L. Rissland, 1997. *Integrating ir and cbr to locate relevant texts and passages.* En *DEXA '97: Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, página 206. IEEE Computer Society, Washington, DC, USA. ISBN 0-8186-8147-0.

J. de Bruijn, R. Lara, S. Arroyo, J. M. Gomez, H. Sung-Kook, y D. Fensel, 2005. *A Unified Semantic Web Services Architecture based on WSMF and UPML.* International Journal on Web Engineering Technology, special issue on Semantic Web.

J. de Bruijn, H. Lausen, A. Polleres, y D. Fensel, 2006. *The web service modeling language wsml: An overview.* En Y. Sure y J. Domingue, editores, *ESWC*, tomo 4011 de *Lecture Notes in Computer Science*, páginas 590–604. Springer. ISBN 3-540-34544-2.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, y R. Harshman, 1999. *Indexing by latent semantic analysis.* Journal of the American Society for Information Science, 41(6):391–407. ISSN 1097-4571.

S. J. Delany y D. Bridge, 2006. *Feature-based and feature-free textual CBR: A comparison in spam filtering.* En *Procs. of the 17th Irish Conference on Artificial Intelligence and Cognitive Science*, páginas 244–253. Belfast, Northern Ireland.

S. J. Delany y P. Cunningham, 2004. *An analysis of case-base editing in a spam filtering system.* En Funk y González-Calero (2004), páginas 128–141.

B. Díaz-Agudo, 2002. *Una aproximación ontológica al desarrollo de sistemas de razonamiento basado en casos.* Tesis Doctoral, Universidad Complutense de Madrid.

B. Díaz-Agudo y P. A. González-Calero, 2000. *An architecture for knowledge intensive CBR systems.* En Blanzieri y Portinale (2000), páginas 37–48.

B. Díaz-Agudo y P. A. González-Calero, 2001a. *Formal concept analysis as a support technique for CBR.* En *Knowledge-Based Systems, 14 (3-4), June,.* Elsevier. (ISSN:0950-7051), pp. 163-172.

B. Díaz-Agudo y P. A. González-Calero, 2001b. *Knowledge intensive cbr made affordable.* En Weber y Gresse von Wangenheim (2001).

B. Díaz-Agudo y P. A. González-Calero, 2001c. *Knowledge intensive CBR through ontologies.* En B. Lees, editor, *Procs. of the 6th UK Workshop on Case-Based Reasoning, UKCBR 2001.* CMS Press,University of Greenwich.

B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García, y A. Sánchez, 2007a. *Building cbr systems with jcolibri.* Special Issue on Experimental Software and Toolkits of the Journal Science of Computer Programming, 69(1-3):68–75. ISSN 0167-6423.

B. Díaz-Agudo, G. Jiménez-Díaz, y J. A. Recio-García, 2008a. *How to teach semantic web?: a project-based approach.* En J. Amillo, C. Laxer, E. M. Ruiz, y A. Young, editores, *ITiCSE*, páginas 98–102. ACM. ISBN 978-1-60558-078-4.

B. Díaz-Agudo, E. Plaza, J. A. Recio-García, y J. L. Arcos, 2008b. *Noticeably new: Case reuse in originality-driven tasks.* En Althoff et al. (2008), páginas 165–179.

B. Díaz-Agudo, J. A. Recio-García, y P. A. González-Calero, 2007b. *Natural language queries in cbr systems.* En *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), 2007, Patras, Greece, Volume 2*, páginas 468–472. IEEE Computer Society. ISSN 1082-3409.

M. E. Fayad y D. C. Schmidt, 1997. *Object-oriented application frameworks - introduction.* Communications of the ACM, 40(10):32–38.

D. Fensel, R. Groenboom, y G. R. R. de Lavalette, 1998. *Modal Change Logic (MCL): Specifying the reasoning of knowledge-based systems.* Data & Knowledge Engineering, 26(3):243–269.

D. Fensel, E. Motta, F. van Harmelen, V. R. Benjamins, M. Crubézy, S. Decker, M. Gaspari, R. Groenboom, W. E. Grosso, M. A. Musen, E. Plaza, G. Schreiber, R. Studer, y B. J. Wielinga, 2003. *The Unified*

Problem-Solving Method Development Language UPML. Knowledge and Information Systems, 5(1):83–131. ISSN 0219-1377.

D. E. Forsythe y B. G. Buchanan, 1993. *Knowledge acquisition for expert systems: some pitfalls and suggestions.* Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems, páginas 117–124.

P. Funk y P. A. González-Calero, editores, 2004. *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings*, tomo 3155 de *Lecture Notes in Computer Science.* Springer. ISBN 3-540-22882-9.

B. G. y P. Denommee, 1999. *Documenting Frameworks to Assist Application Developers.* John Wiley and Sons.

E. Gamma, R. Helm, R. Johnson, y J. Vlissides, 1995. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201633612.

GATE, 2008. *A general architecture for text engineering. http://www.gate.ac.uk/.*

Y. Gil, E. Motta, V. R. Benjamins, y M. A. Musen, editores, 2005. *The SemanticWeb, 4th International SemanticWeb Conference, ISWC 2005*, tomo 3729 de *Lecture Notes in Computer Science.* Springer, Galway, Ireland. ISBN 3-540-29754-5.

A. K. Goel y S. Craw, 2006. *Design, innovation and case-based reasoning.* The Knowledge Engineering Review, 20(03):271–276.

M. H. Göker, R. J. Howlett, y J. E. Price, 2006. *Case-based reasoning for diagnosis applications.* The Knowledge Engineering Review, 20(03):277–281.

M. H. Göker, T. Roth-Berghofer, R. Bergmann, T. Pantleon, R. Traphöner, S. Wess, y W. Wilke, 1998. *The development of homer: A case-based cad/cam help-desk support tool.* En Smyth y Cunningham (1998), páginas 346–357.

G. Golub y W. Kahan, 1965. *Calculating the singular values and pseudo-inverse of a matrix.* Journal of the Society for Industrial and Applied Mathematics, 2(2):205–224.

H. Gómez-Gauchía, B. Díaz-Agudo, y P. A. González-Calero, 2005. *Cobber, towards an affective conversational ki-cbr framework.* En B. Prasad, editor, *IICAI*, páginas 1804–1820. IICAI. ISBN 0-9727412-1-6.

A. Gómez-Pérez, R. González-Cabero, y M. Lama, 2004. *Ode sws: A framework for designing and composing semantic web services.* IEEE Intelligent Systems, 19(4):24–31.

P. González-Calero, 1997. *Aplicación de técnicas basadas en conocimiento como soporte a la reutilización en bibliotecas orientadas a objetos.* Tesis Doctoral, Universidad Complutense de Madrid.

P. A. González-Calero, B. Díaz-Agudo, J. A. Recio-García, y J. J. Bello-Tomás, 2006. *Authoring tools in jcolibri.* Expert Update, 8(3):16–21. ISSN 1465-4091.

P. A. González-Calero, M. Gómez-Albarrán, y B. Díaz-Agudo, 1999a. *Applying DLs for Retrieval in Case-Based Reasoning.* En *Procs. of the 1999 Description Logics Workshop (Dl '99).* Linkopings universitet, Sweden.

P. A. González-Calero, M. Gómez-Albarrán, y B. Díaz-Agudo, 1999b. *A Substitution-based Adaptation Model.* En *Challenges for Case-Based Reasoning - Proc. of the ICCBR'99 Workshops.* Univ. of Kaiserslautern.

T. R. Gruber, 1993. *A translation approach to portable ontology specifications.* Knowledge Acquisition, 5(2):199–220. ISSN 0001-2998.

M. Gu y A. Aamodt, 2005. *A knowledge-intensive method for conversational cbr.* En Muñoz-Avila y Ricci (2005), páginas 296–311.

N. Guarino, 1997. *Understanding, building and using ontologies. a commentary to using explicit ontologies in kbs development, by van heijst, schreiber, and wielinga.* International Journal of Human and Computer Studies, 46(2/3):293–310.

J. L. Guigues y V. Duquenne, 1986. *Familles minimales dímplications informatives resultant dún tableau de données binaires. Math. Sci. Humanies 95, 1986, 5-18.*

K. Gupta, 1998. *Knowledge-based system for troubleshooting complex equipment.* International Journal of Information and Computing Science, 1:29–41.

K. M. Gupta, 2001. *Taxonomic conversational case-based reasoning.* En Aha y Watson (2001), páginas 219–233.

K. M. Gupta y D. W. Aha, 2004. *Towards acquiring case indexing taxonomies from text.* En V. Barr y Z. Markov, editores, *FLAIRS Conference*, páginas 307–315. AAAI Press. ISBN 1-57735-201-7.

K. M. Gupta, D. W. Aha, y N. Sandhu, 2002. *Exploiting taxonomic and causal relations in conversational case retrieval.* En Craw y Preece (2002), páginas 133–147.

V. Haarslev y R. Möller, 2001. *Description of the racer system and its applications.* En *Working Notes of the 2001 International Description Logics Workshop (DL-2001), Stanford, CA, USA, August 1-3, 2001.*

K. J. Hammond, 1989. *Case-based planning: viewing planning as a memory task.* Academic Press Professional, Inc., San Diego, CA, USA. ISBN 0-12-322060-2.

S. Handschuh y S. Staab, editores, 2003. *Annotation for the semantic Web.* IOS Press. ISBN 1-58603-345.

E. Hatcher y O. Gospodnetic, 2004. *Lucene in Action (In Action series).* Manning Publications Co., Greenwich, CT, USA. ISBN 1932394281.

R. Heider, 1996. *Troubleshooting cfm 56-3 engines for the boeing 737 - using cbr and data-mining.* En Smith y Faltings (1996), páginas 512–518.

D. Hennessy y D. Hinkle, 1992. *Applying case-based reasoning to autoclave loading.* IEEE Expert: Intelligent Systems and Their Applications, 7(5):21–26. ISSN 0885-9000.

J. L. Herlocker, J. A. Konstan, A. Borchers, y J. Riedl, 1999. *An algorithmic framework for performing collaborative filtering.* En *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, páginas 230–237. ACM, New York, NY, USA. ISBN 1-58113-096-1.

M. Jaczynski y B. Trousse, 1998. *An object-oriented framework for the design and the implementation of case-based reasoners.* En *Proceedings of the 6th German Workshop on Case-Based Reasoning.*

R. E. Johnson, 1997. *Frameworks = (components + patterns).* Communications of the ACM, 40(10):39–42.

J. Kelleher y D. Bridge, 2004. *An accurate and scalable collaborative recommender.* Artificial Intelligence Review, 21(3-4):193–213. ISSN 0269-2821.

E. Keogh, S. Lonardi, y C. Ratanamahatana, 2004. *Towards parameter-free data mining.* En *Procs. of the 10th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining*, páginas 206–215. New York, NY, USA.

M. Kifer, G. Lausen, y J. Wu, 1995. *Logical foundations of object-oriented and frame-based languages.* Journal of the ACM, 42(4):741–843.

J. Koehler, 1994. *An Application of Terminological Logics to CBR.* En J. Doyle, E. Sandewall, y P. Torasso, editores, *KR'94: Principles of Knowledge Representation and Reasoning*, páginas 351–362. Morgan Kaufmann.

R. Kohavi, D. Sommerfield, y J. Dougherty, 1996. *Data mining using MLC++: A machine learning library in C++.* En *Tools with Artificial Intelligence*, páginas 234–245. IEEE Computer Society Press.

J. Kolodner, 1993. *Case-based Reasoning.* Morgan Kaufmann.

J. Kolodner, 1996. *Making the implicit explicit: Clarifying the principles of case-based reasoning.* En Leake (1996), páginas 349–370.

R. Kosala y H. Blockeel, 2000. *Web mining research: a survey.* SIGKDD Explor. Newsl., 2(1):1–15. ISSN 1931-0145.

P. Koton, 1988. *Reasoning about evidence in causal explanations.* En *Proceedings of the 7th National Conference on Artificial Intelligence*, páginas 256–263. AAAI Press / The MIT Press.

N. Kushmerick, D. Weld, y R. Doorenbos, 1997. *Wrapper induction for information extraction.* En *Fifteenth International Joint Conference on Artificial Intelligence, IJCAI'97.* Morgan Kaufmann, Nagoya, Japan.

L. Lamontagne y G. Lapalme, 2004. *Textual reuse for email response.* En Funk y González-Calero (2004), páginas 242–255.

D. B. Leake, 1992. *Evaluating Explanations: A Content Theory.* Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA. ISBN 0805810641.

D. B. Leake, 1996. *Case-Based Reasoning: Experiences, Lessons and Future Directions.* MIT Press, Cambridge, MA, USA. ISBN 026262110X.

M. Lenz, 1998. *Defining knowledge layers for textual case-based reasoning.* En Smyth y Cunningham (1998), páginas 298–309.

M. Lenz y H. Burkhard, 1997. *CBR for document retrieval—the FAllQ project.* En D. B. Leake y E. Plaza, editores, *Case-Based Reasoning Research and Development (ICCBR-97), Lecture Notes in Artificial Intelligence No. 1266*, tomo 1266 de *Lecture Notes in Computer Science*, páginas 84–93. Springer-Verlag, Berlin. ISBN 3-540-63233-6.

B. López y E. Plaza, 1997. *Case-based learning of plans and goal states in medical diagnosis.* Artificial Intelligence in Medicine, 9(1):29–60.

R. Lopez de Mantaras y E. Plaza, 1997. *Case-based reasoning: An overview.* AI Communications, 10(1):21–29.

M. Luxenburguer, 1991. *Implications partielles dans un contexte.* Mathématiques, Informatique et Sciences Humaines, 113(29):35–55.

R. Mac Gregor y R. Bates, 1987. *The loom knowledge representation language.* ISI Reprint Series ISI/RS-87-188, University of Southern California.

C. D. Manning y H. Schütze, 1999. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts.

D. Matuschek y K. Jantke, 1997. *Axiomatic characterizations of structural similarity for case-based reasoning.* En D. Danke, editor, *Proceedings of the Florida AI Research Symosium (FLAIRS–97).*, páginas 432–436. IA Research Society.

J. McDermott, 1988. *Preliminary steps towards a taxonomy of problem-solving methods.* En S. Marcus, editor, *Automating Knowledge Acquisition for Knowledge-Based Systems.* Kluwer Academic Publishers, Boston.

L. McGinty y B. Smyth, 2002. *Comparison-based recommendation.* En Craw y Preece (2002), páginas 575–589.

E. McKenna y B. Smyth, 2000. *Competence-guided case-base editing techniques.* En Blanzieri y Portinale (2000), páginas 186–197.

D. McSherry, 2001. *Interactive case-based reasoning in sequential diagnosis.* Applied Intelligence, 14(1):65–76.

D. McSherry, 2002a. *Diversity-conscious retrieval.* En Craw y Preece (2002), páginas 219–233.

D. McSherry, 2002b. *Mixed-initiative dialogue in case-based reasoning.* En Aha (2002), páginas 1–8.

D. McSherry, 2003. *Similarity and compromise.* En Ashley y Bridge (2003), páginas 291–305.

A. Mille, B. Fuchs, y O. Herbeaux, 1996. *A unifying framework for adaptation in case-based reasoning.* En L. John Wiley & Sons, editor, *Procs. ECAI'96. Workshop of Adaptation in Case-Based Reasoning.*

G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, y K. J. Miller, 1990. *Introduction to WordNet: An On-line Lexical Database.* International Journal of Lexicography, 3(4):235–244.

D. S. Moore, 1999. *The Basic Practice of Statistics.* W. H. Freeman & Co., New York, NY, USA. ISBN 0716736276.

H. Muñoz-Avila, D. W. Aha, L. Breslow, y D. S. Nau, 1999. *Hicap: An interactive case-based planning architecture and its application to noncombatant evacuation operations.* En *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence*, páginas 870–875. AAAI Press.

H. Muñoz-Avila, D. W. Aha, D. S. Nau, R. Weber, L. Breslow, y F. Yaman, 2001. *Sin: Integrating case-based reasoning with task decomposition.* En B.Ñebel, editor, *IJCAI*, páginas 999–1004. Morgan Kaufmann. ISBN 1-55860-777-3.

H. Muñoz-Avila y F. Ricci, editores, 2005. *Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23-26, 2005, Proceedings*, tomo 3620 de *Lecture Notes in Computer Science.* Springer. ISBN 3-540-28174-6.

I. Muslea, S. Minton, y C. Knoblock, 1998. *Wrapper induction for semistructured, web-based information sources.* En *Proceedings of the Conference on Automatic Learning and Discovery CONALD-98.*

A.Ñapoli, J. Lieber, y R. Courien, 1996. *Classification-Based Problem Solving in CBR.* En Smith y Faltings (1996), páginas 295–308.

D.Ñavinchandra, 1991. *Exploration and innovation in design: towards a computational model.* Springer-Verlag New York, Inc., New York, NY, USA. ISBN 0-387-97481-4.

A.Ñewel, 1982. *The knowledge level.* Artificial Intelligence, 18:87–127.

K.Ñigam, 1999. *Using maximum entropy for text classification.* En *In IJCAI-99 Workshop on Machine Learning for Information Filtering*, páginas 61–67.

OpenNLP, 2008. *http://opennlp.sourceforge.net/.*

S. Osinski, J. Stefanowski, y D. Weiss, 2004. *Lingo: Search results clustering algorithm based on singular value decomposition.* En M. A. Klopotek, S. T. Wierzchon, y K. Trojanowski, editores, *Intelligent Information Systems*, Advances in Soft Computing, páginas 359–368. Springer. ISBN 3-540-21331-7.

D. W. Patterson, N. Rooney, V. Dobrynin, y M. Galushka, 2005. *Sophia: A novel approach for textual case-based reasoning.* En L. P. Kaelbling y A. Saffiotti, editores, *IJCAI*, páginas 15–20. Professional Book Center. ISBN 0938075934.

M. Pearce, A. K. Goel, J. L. Kolodner, C. Zimring, L. Sentosa, y R. Billington, 1992. *Case-based design support: A case study in architectural design.* IEEE Expert: Intelligent Systems and Their Applications, 7(5):14–20. ISSN 0885-9000.

E. Plaza y J. L. Arcos, 2002. *Constructive adaptation.* En Craw y Preece (2002), páginas 306–320.

V. A. Propp, 1968. *Morphology of the Folktale.* Publications of the American Folklore Society. University of Texas Press. ISBN 0292783760.

L. Purvis y P. Pu, 1998. *Composer: A case-based reasoning system for engineering design.* Robotica, 16(3):285–295. ISSN 0263-5747.

A. Ratnaparkhi, 1998. *Maximum entropy models for natural language ambiguity resolution.* Ph.D. thesis, University of Pennsylvania, Philadelphia.

J. A. Recio-García, D. G. Bridge, B. Díaz-Agudo, y P. A. González-Calero, 2008a. *CBR for CBR: A Case-Based Template Recommender System for Building Case-Based Systems.* En Althoff et al. (2008), páginas 459–473.

J. A. Recio-García, B. Díaz-Agudo, D. Bridge, y P. A. González-Calero, 2007a. *Semantic templates for designing recommender systems.* En M. Petridis, editor, *Proceedings of the 12th UK Workshop on Case-Based Reasoning*, páginas 64–75. CMS Press, University of Greenwich. ISBN 987-1-904521-48-8.

J. A. Recio-García, B. Díaz-Agudo, J. M. Gómez-Berbis, y P. A. González-Calero, 2008b. *The wings of jcolibri: A cbr architecture for semantic web services.* International Journal of Web Services Practices (IJWSP). Selected papers from the International Conference on Next Generation Web Services Practices Conference (NWeSP'07), 3(1):35–40. ISSN 1738-6535.

J. A. Recio-García, B. Díaz-Agudo, M. A. Gómez-Martín, y N. Wiratunga, 2005a. *Extending jCOLIBRI for textual CBR.* En Muñoz-Avila y Ricci (2005), páginas 421–435.

J. A. Recio-García, B. Díaz-Agudo, y P. A. González-Calero, 2005b. *A distributed cbr framework through semantic web services.* En M. Bramer, F. Coenen, y T. Allen, editores, *Research and Development in Intelligent Systems XXII (Proc. of theTwenty-fith SGAI Int. Conf. on Innovative Techniques and Applicationsof Artificial Intelligence,AI 2005)*, páginas 88–101. Springer. ISBN 1-84628-225-X.

J. A. Recio-García, B. Díaz-Agudo, y P. A. González-Calero, 2007b. *jCOLIBRI 2 Tutorial.* Informe Técnico IT/2007/02, Departamento de Ingeniería del Software e Inteligencia Artificial. Universidad Complutense de Madrid. Http://gaia.fdi.ucm.es/projects/jcolibri/jcolibri2/docs.html ISBN 978-84-691-6204-0.

J. A. Recio-García, B. Díaz-Agudo, y P. A. González-Calero, 2007c. *Textual cbr in jcolibri: From retrieval to reuse.* En D. C. Wilson y D. Khemani, editores, *Proceedings of the ICCBR 2007 Workshop on Textual Case-Based Reasoning: Beyond Retrieval.*, páginas 217–226. ISBN 978-1-85923-22-4.

J. A. Recio-García, B. Díaz-Agudo, y P. A. González-Calero, 2008c. *Prototyping Recommender Systems in jCOLIBRI.* En *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, páginas 243–250. ACM, New York, NY, USA. ISBN 978-1-60558-093-7.

J. A. Recio-García, B. Díaz-Agudo, y P. A. González-Calero, 2009. *Semantic Templates for Case-Based Reasoning Systems.* The Knowledge Engineering Review. Special Issue on Ontologies Conceptualization and Epistemology for Software and System Engineering. ISSN 0269-8889. Aceptado, pendiente de publicación.

J. A. Recio-García, B. Díaz-Agudo, P. A. González-Calero, y A. Sánchez-Ruiz-Granados, 2006a. *Ontology based cbr with jcolibri.* En R. Ellis, T. Allen, y A. Tuson, editores, *Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, páginas 149–162. Springer, Cambridge, United Kingdom. ISBN 1-84628-665-4.

J. A. Recio-García, B. Díaz-Agudo, A. Sánchez, y P. A. González-Calero, 2006b. *Lessons learnt in the development of a cbr framework.* En M. Petridis, editor, *Proccedings of the 11th UK Workshop on Case Based Reasoning*, páginas 60–71. CMS Press,University of Greenwich. ISBN 987-1-904521-39-6.

J. A. Recio-García, M. A. Gómez-Martín, B. Díaz-Agudo, y P. A. González-Calero, 2006c. *Improving annotation in the semantic web and case authoring in textual cbr.* En T. R. Roth-Berghofer, M. H. Göker, y H. A. Güvenir, editores, *Advances in Case-Based Reasoning, 8th European Conference, ECCBR'06*, tomo 4106 de *Lecture Notes in Artificial Intelligence, subseries of LNCS*, páginas 226–240. Springer, Fethiye, Turkey. ISBN 978-3-540-36843-4. ISSN 0302-9743.

J. A. Recio-García, A. Sánchez, B. Díaz-Agudo, y P. A. González-Calero, 2005c. *jcolibri 1.0 in a nutshell. a software tool for designing cbr systems.* En M. Petridis, editor, *Proccedings of the 10th UK Workshop on Case Based Reasoning*, páginas 20–28. CMS Press,University of Greenwich. ISBN 1-904521-30-4.

F. Ricci y A. Venturini, 2002. *Intelligent travel recommendation.* En Aha (2002), páginas 32–33.

M. M. Richter, 1998. *Introduction.* En *Case-Based Reasoning Technology, From Foundations to Applications*, páginas 1–16. Springer-Verlag, London, UK. ISBN 3-540-64572-1.

M. Riichiro, V. Johan, y I. Mitsuru, 1995. *Task ontology for reuse of problem solving knowledge.* En *Knowledge Building & Knowledge Sharing (KB&KS'95) (2nd International Conference on Very Large-Scale Knowledge Bases).*

E. L. Rissland y J. J. Daniels, 1996. *The synergistic application of cbr to ir.* Artificial Intelligence Review, 10(5-6):441–475. ISSN 0269-2821.

E. L. Rissland y D. B. Skalak, 1991. *Cabaret: Rule interpretation in a hybrid architecture.* International Journal of Man-Machine Studies, 34(6):839–887.

E. L. Rissland, E. M. Valcarce, y K. D. Ashley, 1984. *Explaining and arguing with examples.* En *Proceedings of the National Conference on Artificial Intelligence.*, páginas 288–294. AAAI Press.

D. Roberts y R. E. Johnson, 1997. *Evolving frameworks: A pattern language for developing object-oriented frameworks.* En *Pattern Languages of Program Design 3.* Addison Wesley.

D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, y D. Fensel, 2005. *Web service modeling ontology.* Applied Ontology, 1(1):77–106.

C. Sable, K. McKeown, y K. W. Church, 2002. *NLP found helpful (at least for one text categorization task).* En *Procs. of the ACL-02 conference on Empirical methods in NLP*, páginas 172–179. Association for Computational Linguistics, USA.

S. Salotti y V. Ventos, 1998. *Study and Formalization of a CBR System using a Description Logic.* En Smyth y Cunningham (1998), páginas 286–301.

G. Salton, A. Wong, y C. S. Yang, 1975. *A vector space model for automatic indexing.* Communications of the ACM, 18(11):613–620. ISSN 0001-0782.

R. C. Schank y R. P. Abelson, 1977. *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures.* L. Erlbaum, Hillsdale, NJ.

A. Schappert, P. Sommerlad, y W. Pree, 1995. *Automated support for software development with frameworks.* En *Symposium on Software Reusability*, páginas 123–127.

S. Schmitt, P. Dopichaj, y P. Domínguez-Marín, 2002. *Entropy-based vs. similarity-influenced: Attribute selection methods for dialogs tested on different electronic commerce domains.* En Craw y Preece (2002), páginas 380–394.

T. Schreiber, B. J. Wielinga, J. M. Akkermans, W. V. de Velde, y R. de Hoog, 1994. *CommonKADS: A comprehensive methodology for KBS development.* IEEE Expert, 9(6).

S. Schulz, 1999. *CBR-works: A state-of-the-art shell for case-based application building.* En E. Melis, editor, *Proceedings of the 7th German Workshop on Case-Based Reasoning, GWCBR'99, Würzburg, Germany*, páginas 166–175. University of Würzburg.

J. Schumacher, 2002. *empolis orenge – an open platform for knowledge management applications.* En *1st German Workshop on Experience Management.*

H. Shimazu, 2002. *ExpertClerk: A Conversational Case-Based Reasoning Tool for Developing Salesclerk Agents in E-Commerce Webshops.* Artificial Intelligence Review, 18(3-4):223–244. ISSN 0269-2821.

F. Shull, F. Lanubile, y V. R. Basili, 2000. *Investigating reading techniques for object-oriented framework learning.* IEEE Transactions on Software Engineering, 26(11):1101–1118.

R. Simpson, 1985. *A Computer Model of Case-based Reasoning in Problem-solving: An Investigation in the Domain of Dispute Mediation.* Tesis Doctoral, School of Information and Computer Science, Georgia Institute of Technology. Georgia Institute of Technology, Technical Report GIT-ICS-85/18.

E. Sirin y B. Parsia, 2004. *Pellet: An OWL DL Reasoner.* En V. Haarslev y R. Möller, editores, *Description Logics*, tomo 104 de *CEUR Workshop Proceedings.* CEUR-WS.org.

E. E. Smith, N. Adams, y D. Schorr, 1978. *Fact retrieval and the paradox of interference.* Cognitive Psychology, 10:438–464.

I. F. C. Smith y B. Faltings, editores, 1996. *Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96, Lausanne, Switzerland, November 14-16, 1996, Proceedings*, tomo 1168 de *Lecture Notes in Computer Science.* Springer. ISBN 3-540-61955-0.

B. Smyth, 2007. *Case-based recommendation.* En P. Brusilovsky, A. Kobsa, y W.Ñejdl, editores, *The Adaptive Web*, tomo 4321 de *Lecture Notes in Computer Science*, páginas 342–376. Springer. ISBN 978-3-540-72078-2.

B. Smyth y P. Cunningham, 1992. *Déjá vu: a hierarchical case-based reasoning system for software design.* En *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, páginas 587–589. John Wiley & Sons, Inc., New York, NY, USA. ISBN 0-471-93608-1.

B. Smyth y P. Cunningham, editores, 1998. *Advances in Case-Based Reasoning, 4th European Workshop, EWCBR-98, Dublin, Ireland, September 1998, Proceedings*, tomo 1488 de *Lecture Notes in Computer Science.* Springer. ISBN 3-540-64990-5.

B. Smyth y M. T. Keane, 1995. *Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems.* En *14 th International Joint Conference on Artificial Intelligence (IJCAI'95)*, páginas 377–383. Morgan Kaufmann.

B. Smyth y P. McClave, 2001. *Similarity vs. diversity.* En Aha y Watson (2001), páginas 347–361.

B. Smyth y L. McGinty, 2003. *The power of suggestion.* En G. Gottlob y T. Walsh, editores, *IJCAI*, páginas 127–132. Morgan Kaufmann.

SnowBall, 2008. *http://snowball.tartarus.org/*.

S. Sparks, K. Benner, y C. Faris, 1996. *Managing object-oriented framework reuse*. Computer, 29(9):52–61. ISSN 0018-9162.

A. Stahl y T. Roth-Berghofer, 2008. *Rapid prototyping of cbr applications with the open source tool mycbr*. En K.-D. Althoff y R. Bergmann, editores, *Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008 (in press)*, LNCS. Springer.

R. Studer, V. R. Benjamins, y D. Fensel, 1998. *Knowledge engineering: Principles and methods*. Data & Knowledge Engineering, 25(1-2):161–197.

K. Sycara, 1987. *Resolving Adversarial Conflicts: An Approach Integrating Case-based and Analytic Methods*. Tesis Doctoral, School of Information and Computer Science, Georgia Institute of Technology. Georgia Institute of Technology, Technical Report GIT-ICS-87/26.

The OWL Services Coalition, 2004. *OWL-S: Semantic Markup for Web Services*. http://www.daml.org/services/owl-s/1.1/overview/.

I. Tomek, 1976. *An Experiment with the Edited Nearest-Neighor Rule*. IEEE Transactions on Systems, Man, and Cybernetics, 6(6):448–452. ISSN 1083-4427.

D. Tsarkov y I. Horrocks, 2006. *Description logic reasoner: System description*. En U. Furbach y N. Shankar, editores, *IJCAR*, tomo 4130 de *Lecture Notes in Computer Science*, páginas 292–297. Springer. ISBN 3-540-37187-7.

E. Tulving, 1972. *Episodic and semantic memory*. Organisation of Memory, páginas 381–403. From:Tulving and Donaldson, Organization of Memory, chap 10.

M. Uschold y M. Gruninger, 1996. *Ontologies: Principles, methods and applications*. The Knowledge Engineering Review, 11(2):93–136.

B. Van de Velde, 1994. *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components*. IOS Press, Amsterdam, The Netherlands, The Netherlands. ISBN 90-5199-164-9.

G. van Heijst, A. T. Schreiber, y B. J. Wielinga, 1997. *Using explicit ontologies in kbs development*. International Journal of Human and Computer Studies, 46(2-3):183–292. ISSN 1071-5819.

I. Vollrath, W. Wilke, y R. Bergmann, 1998. *Case-based reasoning support for online catalog sales*. IEEE Internet Computing, 2(4):47–54. ISSN 1089-7801.

E. M. Voorhees, 1993. *Using wordnet to disambiguate word senses for text retrieval*. En *Procs. of the 16th Int. ACM SIGIR conference on Research and development in Information Retrieval*, páginas 171–180. ACM Press, New York,USA. ISBN 0-89791-605-0.

W3C, 2004a. *Owl web ontology language guide*. http://www.w3.org/TR/owl-guide/. World Wide Web Consortium.

W3C, 2004b. *Owl web ontology language overview.* http://www.w3.org/TR/owl-features/. World Wide Web Consortium.

W3C, 2004c. *Rdf primer.* http://www.w3.org/TR/rdf-primer/. World Wide Web Consortium.

W3C, 2004d. *Rdf vocabulary description language 1.0: Rdf schema.* http://www.w3.org/TR/rdf-schema/. World Wide Web Consortium.

I. Watson, 1997. *Applying Case-Based Reasoning.* Morgan Kaufmann.

I. Watson y F. Marir, 1994. *Case-based reasoning: A review.* The Knowledge Engineering Review, 9(4):355–381.

B. Weber, W. Wild, y R. Breu, 2004. *Cbrflow: Enabling adaptive workflow management through conversational case-based reasoning.* En Funk y González-Calero (2004), páginas 434–448.

R. Weber, D. W. Aha, N. Sandhu, y H. Munoz-Avila, 2001. *A textual case-based reasoning framework for knowledge management applications.* En *Proceedings of the 9th German Workshop on Case-Based Reasoning. Shaker Verlag.*

R. Weber y C. Gresse von Wangenheim, editores, 2001. *4th International Conference on Case-Based Reasoning, ICCBR 2001, Vancouver, BC, Canada, July 30 - August 2, 2001, Workshop Proceedings.*

R. Weber, A. Martins, y R. Barcia, 1998. *On legal texts and cases.* Informe técnico, In M. Lenz and K. Ashley eds. Textual Case-Based Reasoning: Papers from the AAAI-98 Workshop, 40-50. AAAI Press.

R. O. Weber, K. D. Ashley, y S. Brüninghaus, 2006. *Textual case-based reasoning.* The Knowledge Engineering Review, 20(03):255–260.

B. Wielinga, A. Schreiber, y J. Breuker, 1992. *Kads: A modelling approach to knowledge engineering.* Knowledge Acquisition, 4(1).

R. Wille, 1982. *Restructuring Lattice Theory: an approach based on hierarchies of concepts.* Ordered Sets.

D. Wilson y S. Bradshaw, 1999. *Cbr textuality.* En *In Proceedings of the Fourth UK Case-Based Reasoning Workshop.*

N. Wiratunga, I. Koychev, y S. Massie, 2004. *Feature selection and generalisation for retrieval of textual cases.* En Funk y González-Calero (2004), páginas 806–820.

I. Witten y E. Frank, 2000a. *Data mining: Practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann.

I. H. Witten y E. Frank, 2000b. *Data mining: practical machine learning tools and techniques with Java implementations.* Morgan Kaufmann, USA. ISBN 1-55860-552-5.

K. E. Wolff, 1993. *A first course in formal concept analysis. How to understand line diagrams.* En F. Faulbaum, editor, *SoftStat, Advanced in Statistical Software*, páginas 429–438. Gustav Fischer Verlag.

L. Xiao, D. Wissmann, M. Brown, y S. Jablonski, 2004. *Information extraction from the web: System and techniques.* Applied Intelligence, 21(2):195–224. ISSN 0924-669X.