Catalogador automático de textos y Recomendador de artículos del Portal de Revistas Electrónicas de la biblioteca UCM

Autores:
Cristina Diz Monje
José Guzmán Garate
Jesús Sánchez García

Director: Belén Díaz Agudo
Proyecto Sistemas Informáticos 2007/2008
Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

RESUMEN

Este trabajo se centra en el proceso de desarrollo de un catalogador automático de textos y un recomendador de libros para la biblioteca de la UCM. Este proyecto ha sido desarrollado utilizando el framework jCOLIBRI2, aplicando técnicas de Inteligencia Artificial como razonamiento basado en casos (CBR), razonamiento textual y formalización lógica del conocimiento. Inicialmente se introducen los conceptos teóricos necesarios para el desarrollo del proyecto como: Sistemas basados en conocimiento, ciclo CBR, CBR textual, arquitectura jCOLIBRI2 y sistemas recomendadores. También se describen las diferentes fases llevadas a cabo durante el proceso de desarrollo de las aplicaciones como la fase de adquisición del conocimiento, desarrollo de prototipos iniciales, realización de experimentos y pruebas, así como los problemas encontrados. Por último, se describe cómo se llevo a cabo la integración del catalogador desarrollado, para su uso en la biblioteca.

PALABRAS CLAVE: Sistemas Recomendadores, Catalogación automática de textos, Sistemas CBR, CBR Textual, jCOLIBRI2, Lucene, Ciclo CBR.

ABSTRACT

This work is focused on the process of development an automatic classification of texts and a book recommender systhem for the library of the UCM. This project has been developed using the framework jCOLIBRI2, applying techniques of Artificial Intelligence like reasoning based on cases (CBR), textual reasoning and logic formalization of knowledge. Firstly, we introduce the necessary theoretical concepts for the development of the project such as: knowledge based systems, CBR cycle, textual CBR, jCOLIBRI2 architecture and recommender systems. We also describe the different phases carried out during the applications development process like the phase of knowledge acquisition, initial prototypes development, experiments and tests, as well as the problems found. Finally, we describe how we carried out the integration of the automatic classification system developed to be used in the library.

KEYWORDS: Recommender Systhems, Automatic Classification of texts, CBR Systems, Textual CBR, jCOLIBRI2, Lucene, CBR cycle.

1. INTRODUCCIÓN Y OBJETIVOS

Este proyecto fue propuesto por el Departamento de Ingeniería del Software e Inteligencia Artificial de la Universidad Complutense de Madrid (UCM). El objetivo del proyecto era desarrollar un sistema de Inteligencia Artificial para la catalogación y recomendación de libros del catálogo general de la biblioteca de la UCM. El proyecto iba a ser realizado en colaboración con el departamento de automatización de la biblioteca de la UCM, si el resultado del proyecto era satisfactorio se implantaría para su uso en la misma.

La biblioteca de la UCM cuenta con muchísimo conocimiento en diferentes formas: libros, artículos, tesis, bases de datos referenciales…Además, la Web de la biblioteca dispone de herramientas que permiten simplificar la búsqueda de material. Uno de los objetivos del proyecto era ampliar y mejorar este tipo de herramientas, como por ejemplo, búsquedas que tienen que ver con el término o los términos que introdujo el usuario, o búsquedas que muestran como resultado una recomendación de libros o documentos porque otros usuarios similares los consultaron previamente. Una Web similar que incluye estos objetivos de recomendación sería la ofrecida por *Amazon* (http://www.amazon.com).

El segundo objetivo era la catalogación automática de textos, construcción de redes semánticas y elaboración de conocimiento de comunidad (recomendadores colaborativos).

En algunas reuniones preliminares, se propusieron tres posibles líneas de colaboración con la biblioteca:

- Catalogación automática de textos (o páginas Web) basada en una base de conocimiento formada por textos previamente catalogados.
- Recomendación de libros: buscador semántico y basado en razonamiento por similitud de los usuarios.
- Ontologías: Redes de conocimiento, extender el diccionario TDC y plantear un mecanismo de exploración gráfica.

Para llevar a cabo estos objetivos, se utilizarían tecnologías CBR, Ontologías y la herramienta jCOLIBRI2 (http://gaia.fdi.ucm.es/grupo/projects/jcolibri2). Para la implementación se usarían herramientas basadas en el lenguaje JAVA.

Una vez fijados los objetivos iniciales del proyecto, nos centramos en la fase del Estado del arte para documentarnos sobre cada una de las distintas metodologías y técnicas necesarias para poder llevar a cabo estos objetivos.

Tras esta fase de documentación previa, se realizó una reunión con el departamento de automatización de la biblioteca de la UCM para llegar a un acuerdo sobre el trabajo de colaboración que íbamos a realizar para ellos. En esta reunión se tomó la decisión de abordar las dos primeras líneas de colaboración propuestas, Catalogación automática de textos y Recomendación de libros, empezando por el desarrollo de la primera de ellas.

2. ESTADO DEL ARTE

2.1. Razonamiento basado en casos (CBR)

El razonamiento basado en casos (CBR, del inglés Case Based Reasoning) es uno de los paradigmas de resolución de problemas más exitosos dentro de la Inteligencia Artificial y actualmente se aplica en multitud de aplicaciones. Este método se engloba dentro de los Sistemas Basados en Conocimiento (SBC), que a su vez es una rama importante dentro de la Inteligencia Artificial. En este apartado empezaremos hablando de los Sistemas Basados en Conocimiento, luego explicaremos el ciclo CBR y por último nos centraremos en el CBR Textual, que es una de las técnicas que hemos utilizado en el desarrollo de nuestro proyecto.

2.1.1 Sistemas Basados en Conocimiento

Los Sistemas Basados en Conocimiento (SBC) forman parte de una rama de la IA en la que se intenta modelar el comportamiento humano experto para resolver un problema. Existen muchas definiciones sobre lo que es un Sistema Basado en Conocimiento, que a su vez han ido evolucionando según se iba investigando sus posibilidades. Actualmente, un SBC se puede definir como un sistema informático que utiliza conocimiento sobre un dominio de aplicación para obtener una solución de un problema en este dominio. El dominio suele ser muy específico y se suele utilizar a un experto del dominio como fuente de conocimiento (también se pueden usar otras fuentes, como fuentes textuales, técnicas de análisis de datos...). En nuestro proyecto el dominio de aplicación son artículos de revistas de la UCM que nos han facilitado los servicios de la biblioteca. En este dominio hemos desarrollado 2 aplicaciones: un catalogador automático y un recomendador. Para el desarrollo de ambas aplicaciones hemos utilizado como motor de inferencia el razonamiento basado en casos.

Es interesante comentar la diferencia entre un sistema basado en conocimiento (de lo que se ocupa la Ingeniería del Conocimiento) y un sistema tradicional (que parte de la Ingeniería del Software). En un SBC la especificación del problema no suele ser completa y el conocimiento se debe adquirir de un experto del dominio. Además, en un SBC siempre existe una clara separación entre la base de conocimiento y los métodos de razonamiento. Las soluciones en este tipo de sistemas suelen ser no deterministas. En cambio, en un sistema tradicional de Ingeniería del Software la especificación suele ser muy completa, los datos y métodos de resolución son conocidos y existe documentación. No existe una separación entre datos y métodos, y las soluciones son rígidas y deterministas. Otra diferencia clara son los tipos de problemas que abarcan estos sistemas habitualmente. En un SBC los problemas suelen ser heurísticos y declarativos, mientras que en un sistema tradicional son más sistemáticos y procedimentales.

Como ya se ha mencionado una de las características más importantes de un SBC es la separación entre el conocimiento y el motor de inferencia. Esta separación permite que se apliquen distintas técnicas de razonamiento a una misma base de conocimiento fácilmente, pudiendo comparar los resultados para un problema concreto. Para obtener el conocimiento se debe extraer de un experto del dominio y representarlo en una base de conocimiento (tareas básicas de la Ingeniería del Conocimiento). Existen distintas técnicas de adquisición de conocimiento y de representación, según sea el tipo de conocimiento (declarativo, procedimental o metaconocimiento). Igualmente hay muchos paradigmas no excluyentes para la resolución de problemas: paradigma lógico, de búsqueda heurística, basado en conocimiento, conexionista, basado en experiencia… Dentro del paradigma basado en experiencia se encuentran el

razonamiento basado en reglas (Rule Based Reasoning), razonamiento basado en casos (Case Based Reasoning) y las ontologías. Nosotros nos centraremos en este paradigma y más concretamente en el CBR.

Con el razonamiento basado en reglas (RBR) se construye el motor de inferencia mediante reglas de la forma "if A then B", estas reglas se disparan cuando se cumplen sus antecedentes A y provocan cambios con sus consecuentes B en la base de conocimiento. Algunas ventajas de este método son la modularidad que proporcionan las reglas, la flexibilidad del orden en que se ejecutan las reglas (con encaje de patrones), y la independencia del lenguaje en la representación de reglas y memoria de trabajo. Además existen varios lenguajes basados en reglas como CLIPS o PROLOG que sirven como herramienta para la generación de reglas. El gran inconveniente que presenta este razonamiento es el cuello de botella que se produce en la adquisición de conocimiento. Se plantea si es posible que el ingeniero de conocimiento extraiga todo el conocimiento de un experto y si es posible formalizar todo ese conocimiento mediante reglas y hechos. Hay muchos tipos de conocimiento que no son expresables en forma de reglas. Otra cuestión problemática es si existe un convenio entre todos los expertos sobre si un modelo es adecuado o no. Como solución a estas y otras cuestiones surge el razonamiento basado en casos (CBR).

El CBR facilita la adquisición de conocimiento en cuanto que sólo hace falta un conjunto de problemas resueltos que sirva de base de conocimiento (base de casos). Hay que tener también unas medidas de similitud entre casos y técnicas de adaptación. Teniendo esto en cuenta la idea es sencilla: dado un problema utilizar el conocimiento adquirido en experiencias anteriores para resolverlo. Así es como actúan las personas en muchas situaciones, por lo que el CBR es el modo natural de razonar de los seres humanos. Además, el CBR facilita el aprendizaje ya que el sistema añade el problema resuelto como un nuevo caso para la base de casos. Esto es interesante ya que los problemas suelen ser recurrentes. Este método es mejor para dominios que no se conocen del todo o que están poco formalizados.

Hay dos tipos básicos de sistemas CBR: de interpretación de situaciones o de resolución de problemas. Los primeros se basan en formular un juicio o clasificar una situación, por ejemplo diagnosticar una enfermedad según unos síntomas. En este tipo de sistemas los casos son fáciles de representar y recuperar, y la adaptación es muy sencilla o nula. Los sistemas CBR de resolución de problemas se basan en aplicar la solución de un problema anterior para obtener la solución al problema actual. Aquí la representación de los casos y la adaptación son más complejas.

2.1.2 Ciclo CBR

El razonamiento basado en casos se explica mediante el ciclo CBR que tiene cuatro fases conocidas como las 4 R's [3]:

- Recuperar: los casos más relevantes para una consulta dada.
- Reutilizar: estos casos para construir una primera solución.
- Revisar: dicha solución adaptándola a la situación actual.
- Recordar: la solución final para añadirla a la base de casos.

En la *figura 2.1* se puede observar la representación gráfica del ciclo CBR:

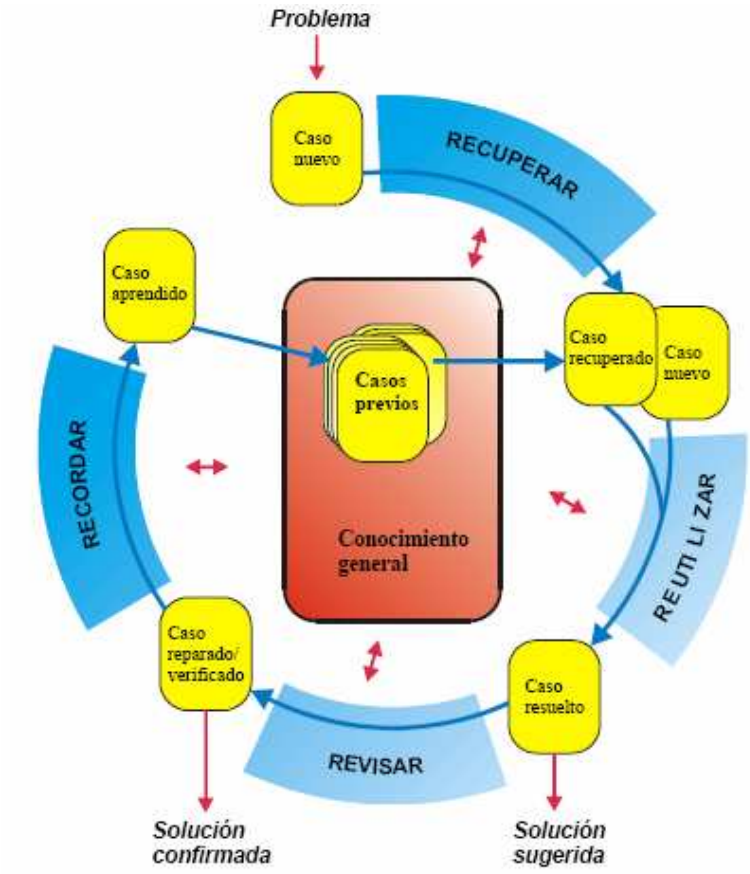


Figura 2.1

En la recuperación de casos se distinguen varias tareas. Primero hay que determinar cuáles son las características importantes (índices) que permiten encontrar los casos relevantes. Se deben tener medidas de similitud local (para cada atributo) y global (con distintos pesos para cada atributo) para poder comparar el nuevo caso con la base de casos. Luego se realiza una búsqueda de los k-vecinos más próximos (k-nearest neighbor) en la base de casos para encontrar los casos más relevantes para resolver el problema. Aquí es importante la organización de la base de casos (lineal, en árbol, mixta, en clusters…) para que la búsqueda sea eficiente y precisa. También cabe distinguir si se desea una recuperación exacta o aproximada de los casos relevantes. Por último, se selecciona el mejor caso.

En la fase reutilizar se usa el conocimiento incluido en el caso recuperado para resolver el problema. Al final de esta etapa se propone una primera solución juntando la información del caso nuevo y el caso recuperado. El usuario puede intervenir en esta fase para adaptar el caso recuperado a la situación actual. Luego viene la fase de revisión, donde se comprueba si la solución propuesta es válida. Esta validación suele hacerse fuera del sistema CBR, ya sea por el usuario, por un experto o por simulación. Por último, una vez aceptada la solución propuesta se debe mostrar el resultado y añadir el caso a la base de casos. Esta fase de aprendizaje es quizá la más interesante del ciclo. Con ella se mejora el sistema cuanto más se use, pues va añadiendo experiencias y corrigiendo errores. Además, así se mejora la eficacia al tener más casos con los que comparar. Si la base de casos aumenta tanto que la eficiencia empeora demasiado, es interesante el uso de algoritmos de mantenimiento que reduzcan la base de casos eliminando casos redundantes.

Este es el ciclo del razonamiento basado en casos pero, ¿qué es un caso? Un caso es una entidad que representa una situación del dominio de aplicación. Básicamente está formado por una descripción y una solución. La descripción es el conjunto de atributos que caracterizan el caso y la solución es el conjunto de atributos que indican cómo resolver el problema. Por ejemplo, en un recomendador de viajes (donde los casos son los distintos tipos de viajes) la descripción del caso podría ser el país de destino, el medio de transporte, las actividades incluidas en el viaje, el hotel… y la solución del caso podría ser el precio y el nombre de la agencia de viajes para contratarlo.

2.1.3 CBR Textual

El CBR Textual es un subapartado del CBR donde las fuentes del conocimiento se dan en formato de texto. El objetivo es utilizar este conocimiento textual para resolver un problema de la misma manera que el CBR, es decir, comparando la consulta inicial en formato textual, con la base de casos (también en formato textual total o parcialmente) y recuperar los más similares. Se mantiene por tanto, el mismo esquema del ciclo CBR: recuperar, reutilizar, revisar y recordar.

Esta rama del razonamiento basado en casos plantea varias cuestiones: cómo hacer similitud entre casos textuales, cómo mapear textos en una representación estructurada de los casos, cómo adaptar casos textuales o cómo generar automáticamente representaciones textuales de los casos. Estas y otras cuestiones se resuelven con técnicas de Recuperación de Información [7] (Information Retrieval, IR). Estas técnicas trabajan con textos no estructurados, consultas en lenguaje natural y realizan una recuperación aproximada. Más formalmente, dado un texto a analizar, el procesamiento del texto se divide en varios procesos secuenciales, que es lo que se conoce como las 7 capas de Lenz [2]:

- Keyword Layer: elimina palabras vacías, hace extracción de raíces…
- Phrase Layer: obtiene expresiones y frases de términos que son dependientes del dominio.
- Thesaurus Layer: obtiene sinónimos e identifica términos relacionados independientemente del dominio.
- Glossary Layer: obtiene sinónimos dependientes del dominio.
- Feature Value Layer: obtiene pares del tipo <atributo, valor>.
- Domain Structure Layer: clasifica el caso según el dominio de aplicación.
- Information Extraction Layer: mejora la estructuración del caso.

En *la figura 2.2* se puede observar un diagrama que muestra algunas de estas técnicas de procesamiento de textos (básicamente las 3 primeras capas de Lenz) relacionado con una base de casos y una consulta. A la izquierda aparecen los documentos de texto que representan la base de casos por lo que los documentos de texto son los casos. A la derecha aparece la consulta del usuario, también en formato de texto. Tanto a la base de casos como a la consulta se le aplican varias de estas técnicas para procesar los textos y obtener la representación correcta para poder calcular la similitud entre cada caso y la consulta. Una vez hecha la similitud se obtienen los documentos (casos) recuperados.

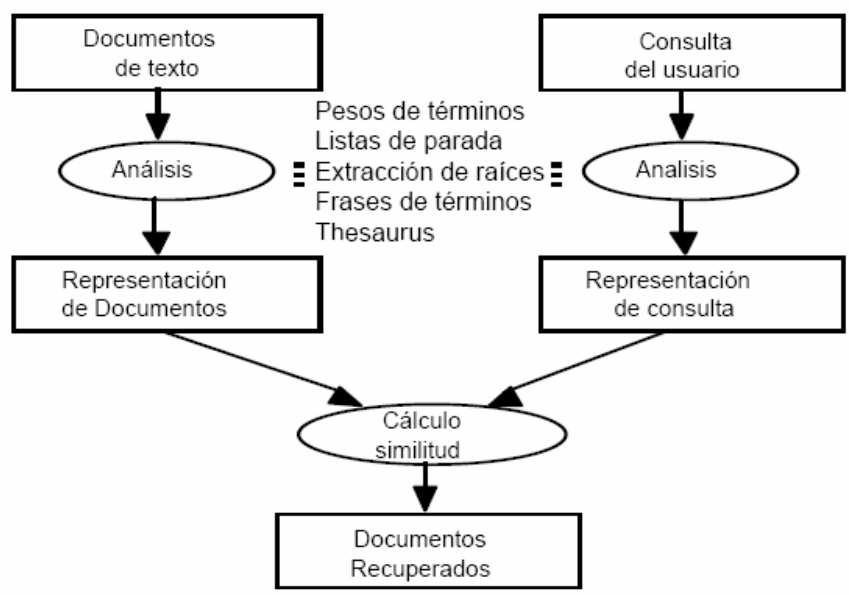


Figura 2.2

Las listas de palabras vacías (o listas de parada) tienen como propósito eliminar las palabras que no son útiles para la representación del caso. Por ejemplo, palabras que aparecen en todos los casos no discriminan, palabras con poco significado, artículos, pronombres… Cada idioma e incluso cada dominio tiene su propio conjunto de palabras vacías. La fase de extracción de raíces pretende obtener un único término de indexación a partir de las diferentes variaciones morfológicas de una palabra. Por ejemplo, las palabras tiene, tuve, tendrán, habremos tenido… tienen todas la raíz "ten".De esta manera se reduce el número de palabras y se simplifica la similitud. En cambio plantea problemas de sobrerradicación (se reduce demasiado y palabras semánticamente distintas obtienen misma raíz) o infrarradicación (no se reduce lo suficiente y palabras semánticamente iguales tienen raíces diferentes) si no se realiza con cuidado. Existen algoritmos implementados como el Stemmer Conservativo UEA-LITE (www.cmp.uea.ac.uk/Research/stemmer) o el Algoritmo de extracción de raíces de Porter [9], que presentan un buen resultado.

La lematización consiste en obtener el infinitivo de cualquier forma verbal. Por ejemplo, de haré, hicimos, hube hecho… se obtienen "hacer". De esta manera no existe ni sobrerradicación ni infrarradicación, pero no es una tarea trivial, pues conlleva un análisis léxico, sintáctico y semántico. También es habitual el uso de recursos externos como WordNet que implemente estas y otras funciones. La tarea de construir frases de términos tiene como objetivo obtener nuevos términos de indexación con un significado más preciso. Por ejemplo, si aparecen juntas las palabras "restaurante" e "italiano" se podría construir el término "restaurante italiano" que para cierto dominio puede resultar un término interesante. En la práctica se ha comprobado como este mecanismo no aumenta la efectividad demasiado. Por último, un mecanismo que si es útil para la extracción de información es el uso de sinónimos de un dominio específico (Thesaurus) y listas de asociación. Lo primero permite identificar términos semánticamente iguales y lo segundo relaciona numéricamente la similitud entre términos. Todas estas fases (o capas) y alguna más, forman lo que formalmente se llama Modelo de Capas de Lenz. Una vez hecho todo esto se puede proceder al cálculo de similitud entre los documentos.

Existen distintos modelos para hacer similitud entre textos. El más sencillo es el modelo booleano que trabaja con palabras índice concatenadas con operadores lógicos, pero este modelo

resulta demasiado estricto. El modelo que más interesante es el Modelo del Espacio Vectorial, en este modelo cada texto se puede interpretar como un vector cuyas componentes son las palabras que lo forman. De esta manera, se trabaja con un espacio vectorial con dimensión n, siendo n el número de palabras distintas en todos los documentos. Con esta idea, la similitud entre textos se puede expresar como el ángulo que forman sus vectores. Si el coseno es pequeño, los vectores están próximos y los textos correspondientes son bastante similares. Además del coseno (cosine coefficient) existen otras medidas como el "dice coefficient", "jaccard coefficient" o "overlap coefficient" [5].

Por último, haremos referencia a las medidas de calidad de un sistema CBR textual más importantes. Por un lado tenemos la eficiencia, que es el tiempo que tarda el sistema en proporcionar una respuesta al usuario. Las funciones de similitud y la representación en memoria de la base de casos son determinantes en la eficiencia. Otra medida importantes es la efectividad del sistema, es decir el porcentaje de éxitos que tiene la aplicación. Para medir la efectividad es necesario tener "a priori" las soluciones de los problemas, para compararlas con las obtenidas por el sistema. Tanto la eficiencia como la eficacia son medidas comunes a cualquier aplicación que se desarrolle. Medidas más propias de un sistema CBR son la precisión y el recall (o exhaustividad). La precisión se define como el cociente entre los documentos relevantes recuperados y el total de documentos recuperados. El recall en cambio, se define como el cociente entre documentos relevantes recuperados y documentos relevantes en toda la base de casos. La clave de estas medidas es cómo determinar si un documento es relevante o no para un problema concreto. Obsérvese que estas medidas son en parte complementarias, si la precisión es muy alta, quizá porque se recuperan muy pocos documentos, entonces el recall tenderá a ser bajo, pues se recuperan pocos documentos relevantes comparando con todos los documentos relevantes en la colección. La situación óptima sería tener tanto precisión como recall altos.

2.2 jCOLIBRI2

Para el desarrollo de sistemas basados en conocimiento existen muchas herramientas ya implementadas que proporcionan una estructura sobre la que implementar un sistema CBR. Una de estas herramientas es la que hemos utilizado para el desarrollo de nuestro proyecto, jCOLIBRI (http://gaia.fdi.ucm.es/grupo/projects/jcolibri). Este framework ha sido desarrollado en el grupo GAIA de la UCM, es de ámbito académico y se utiliza en diversas universidades (Université Pierre et Marie Curie de París, Georgia Institute of Technology en EEUU…).

jCOLIBRI (siglas de "Cases and Ontology Libraries Integration for Building Reasoning Infrastructures") es un sistema que facilita el diseño y enseñanza de los sistemas CBR. Proporciona una arquitectura independiente del dominio en el que se emplee. Actualmente existen dos versiones jCOLIBRI y jCOLIBRI2 disponibles en la página web. La primera versión, desarrollada en LISP, proporciona una interfaz gráfica completa que guía al usuario a diseñar un sistema CBR. Esta versión no está orientada a usuarios desarrolladores y permite realizar una aplicación sin escribir una línea de código. En cambio, jCOLIBRI2 sigue una arquitectura claramente dividida en dos partes: una orientada a diseñadores y otra orientada a desarrolladores. Para la realización de nuestro proyecto hemos utilizado esta segunda versión.

2.2.1 Arquitectura jCOLIBRI2

jCOLIBRI2 [4] es un framework para el desarrollo de aplicaciones CBR. Está formado por un conjunto de paquetes escritos en Java que proporciona muchas herramientas, algoritmos y plantillas para implementar fácilmente un programa que aplica razonamiento basado en casos.

Este framework proporciona una estructura basada en el ciclo CBR (Recuperar, Reutilizar, Revisar y Recordar) para implementar una aplicación CBR. Permite trabajar con bases de casos que pueden venir en distintos formatos a través de conectores. También incluye extensiones para realizar sistemas recomendadores, trabajar con ontologías o algoritmos y métodos para el CBR textual. Además, incluye una aplicación ejecutable con varios ejemplos, un tutorial y documentación para comprender mejor su funcionamiento. En *la figura 2.3* se muestra la arquitectura de jCOLIBRI2:

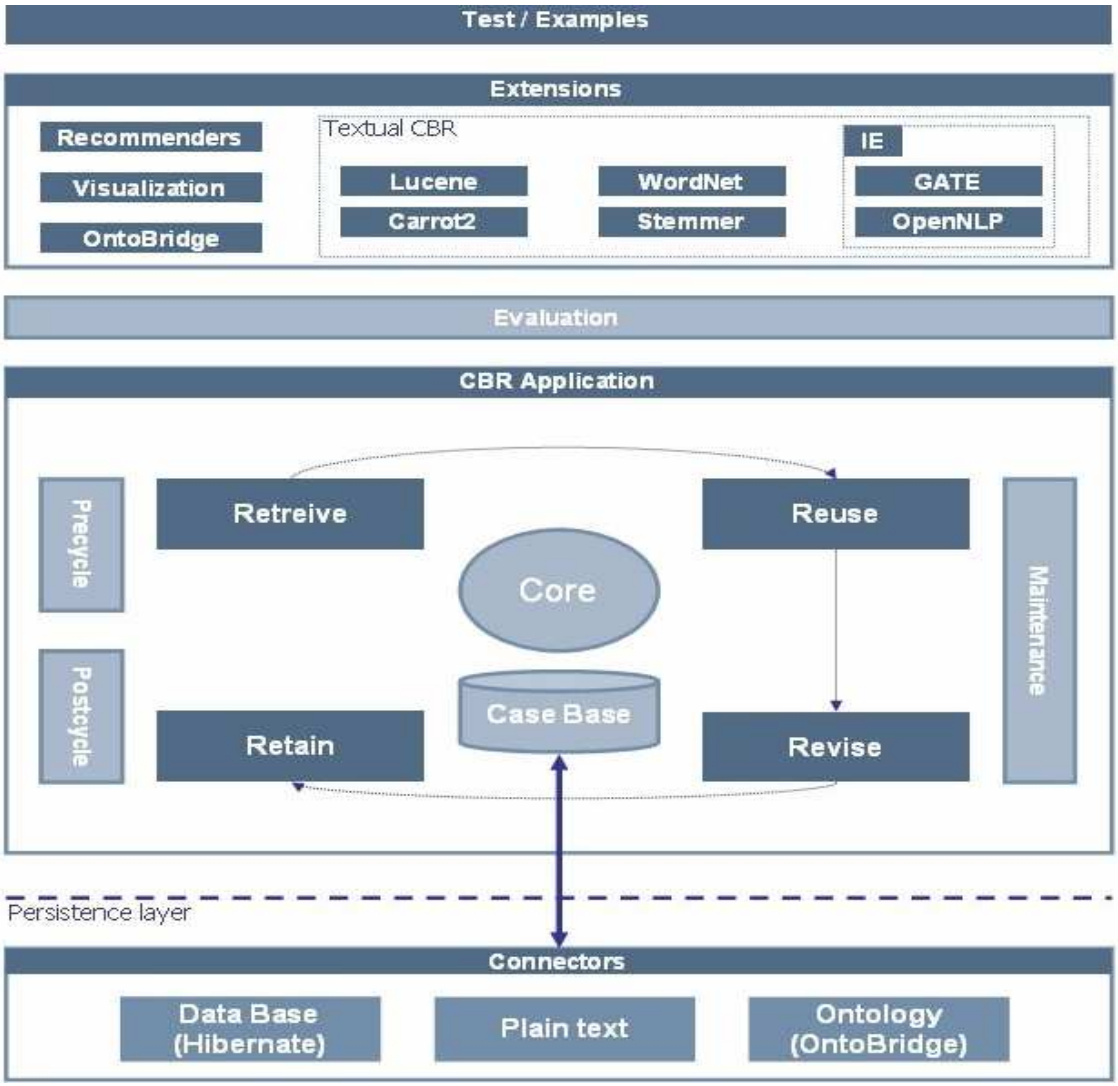


Figura 2.3

El esquema básico de una aplicación CBR usando jCOLIBRI2 se divide en cuatro fases:

- Configuración: configura la persistencia de los casos y prepara la memoria para su utilización.
- Preciclo: crea la base de casos cargando los casos en memoria.
- Ciclo: obtiene la consulta y realiza el ciclo CBR
- Postciclo: cierra los conectores y libera recursos.

Estas fases tienen sus correspondientes métodos (configure(), precycle(), cycle() y postcycle()) que pertenecen a la interfaz StandarCBRAplication, de manera que para diseñar un aplicación CBR se debe crear una clase que implemente dicha interfaz. A continuación explicaremos cada uno de estos métodos.

Lo primero que se debe hacer para desarrollar una aplicación CBR es determinar cómo va a ser la base de casos y el conector para obtener dichos casos. De esto se encarga el método configure(). Hay varios tipos de conectores que básicamente se conectan a 3 tipos de fuentes que sirven de base de conocimiento: conectores a bases de datos, conectores a texto plano y conectores a ontologías. Los primeros conectan a cualquier tipo de base de datos (Access, mySQL, Oracle…) a través de la librería Hibernate, que mapea cada atributo de un caso en una columna de la BBDD. Los segundos leen la información de documentos de texto sin un formato predefinido. Por último, los conectores a ontologías a través del módulo OntoBridge permiten razonar con ontologías escritas en distintos lenguajes. Todos estos conectores se inicializan por medio de un archivo XML. Además, puedes crear tu propio conector simplemente implementando la interfaz Connector. La otra tarea básica del método configure() es elegir una representación para la base de casos, jCOLIBRI2 permite muchos tipos de bases de casos (lineales, indexadas…) según la necesidad de la aplicación.

Después de configurar el conector y la base de casos se procede a cargar los casos en memoria. Esta es la función del método precycle(). El conector lee los casos de la fuente de conocimiento y se crea la base de casos que será parte fundamental del sistema CBR. Ahora ya se puede realizar el ciclo CBR con el método cycle(). Este método obtiene la consulta del usuario y lanza a ejecutar las fases de recuperación, reutilización, revisión y retención. La manera en que obtiene la consulta inicial puede ser muy variada: rellenando un formulario, respondiendo a unas preguntas, escogiendo entre un escaparate de posibilidades, comparando con perfiles de usuarios similares… Para la recuperación de casos relevantes existen muchos algoritmos implementados. El más utilizado es el KNN (k-nearest neighbour) que selecciona los k casos más similares a la consulta. Para hacer la similitud entre casos intervienen funciones de similitud locales (para cada atributo del caso) y una función de similitud global, la cual junta las funciones locales anteriores asociándoles un peso según su importancia. Hay que configurar cada una de estas medidas de similitud según el dominio de aplicación. Al final se devuelven los k casos más relevantes junto a su similitud (valor comprendido entre 0 y 1). Las siguientes fases del ciclo adaptación y revisión son muy dependientes del dominio de la aplicación, y es tarea del diseñador implementar dichas funciones. Para el aprendizaje de los nuevos casos solo se debe llamar al método storecases() del conector para que los incluya en la base de casos.

Por último, una vez terminado el ciclo CBR, se debe llamar al postcycle() para terminar la ejecución de la aplicación. Este método cierra los conectores de manera segura y libera los recursos que se estuviesen utilizando.

Hemos comentado la importancia de la base de casos y su representación, pero sería interesante fijarse en la estructura de un caso genérico para implementarlos correctamente. Un caso está compuesto por una descripción, una solución, una justificación de la solución y un resultado. Estas componentes están formadas por atributos que según la aplicación pueden ser parte de la

descripción o de la solución. El resultado del caso depende de si la solución del mismo es aceptable para resolver el problema o no. En *la figura 2.4* se muestra un diagrama de clases de los casos, bases de casos y conectores. Estas clases constituyen el núcleo (paquete jcolibri.cbrcore en jCOLIBRI2) de una aplicación CBR.

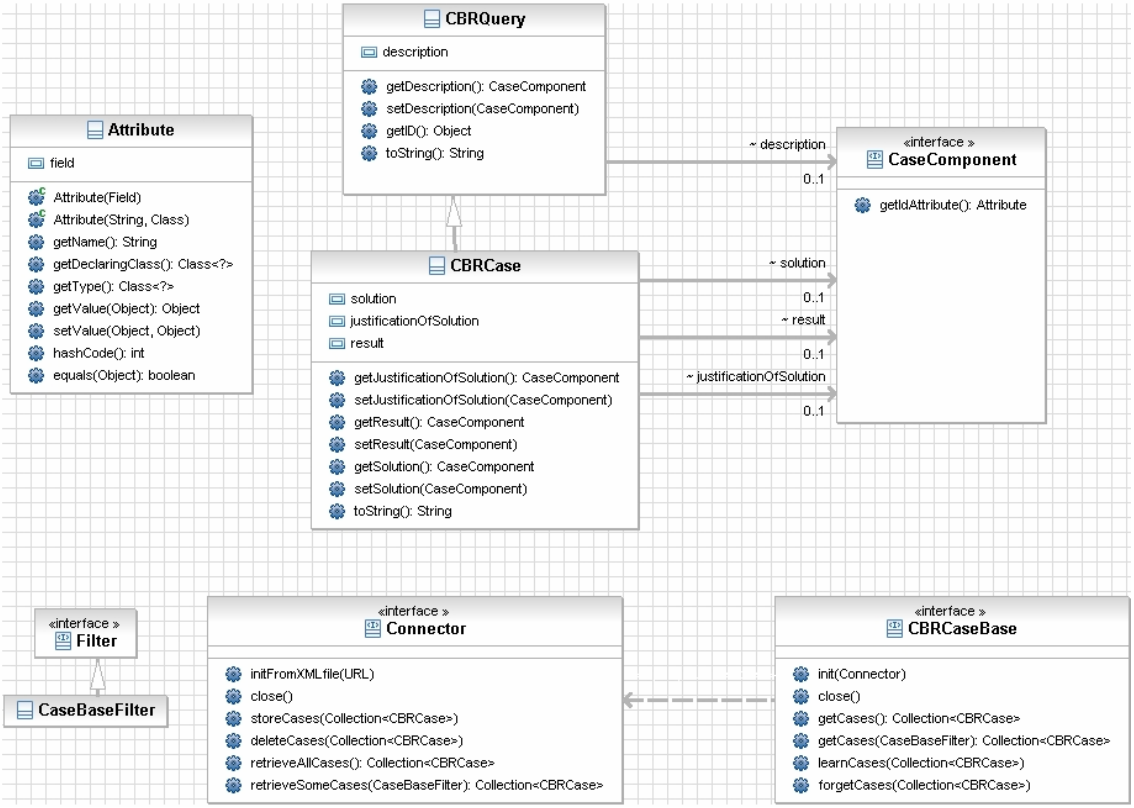


Figura 2.4

2.2.2 CBR Textual con jCOLIBRI2

En este apartado pretendemos explicar brevemente algunos de las extensiones que incluye jCOLIBRI2 para la resolución de problemas CBR textuales.

Como ya se ha mencionado anteriormente, para el tratamiento de textos sin formato (habitualmente en lenguaje natural) es necesario procesar el texto para limpiarlo de palabras vacías, extraer raíces, aplicar librerías de sinónimos, etc. Estas funciones están incluidas en paquetes como OpenNLP [5] (Natural Language Processing) y WordNet [5]. Para la extracción de raíces (stemmers) está el Snowball [5] que permite definir el idioma.

Para la recuperación de casos, un algoritmo ampliamente utilizado y que presenta buenos resultados en un tiempo aceptable es Lucene[7]. Este algoritmo se basa en el Modelo del Espacio Vectorial, que mide la similitud entre casos según el ángulo que forman los vectores que los representan. Otra técnica interesante es la de reducir u organizar la base de casos en conjuntos (clusters) de documentos similares. Esta técnica es utilizada por la extensión Carrot2, que divide la base de casos en clusters y escoge un caso del cluster como representante del mismo, de esta manera se pueden hacer comparaciones sólo con los representantes de cada cluster (reduciendo

considerablemente el tiempo de recuperación de casos), escoger el representante más similar y luego comparar la consulta con el resto de casos del cluster para así afinar más en la recuperación.

2.3. Sistemas Recomendadores

Actualmente los sistemas recomendadores de productos son un tipo de aplicación muy común. Tienen como objetivo presentar un conjunto de elementos (o items) que puedan interesar al usuario. Son aplicaciones en las que el usuario solicita un producto mediante una consulta y el sistema le propone un conjunto de candidatos que cumplan las expectativas del usuario. A simple vista, un recomendador puede parecer lo mismo que un buscador convencional, pero un recomendador tiene mayor capacidad de razonamiento que un buscador. Mientras que un buscador realiza una recuperación exacta, es decir, busca ítems que cumplan estrictamente las restricciones impuestas por el usuario, un recomendador realiza recuperación aproximada, lo que quiere decir que es capaz de proponer ítems que quizá no se ajusten perfectamente a la consulta inicial, pero que a través del motor del inferencia del sistema han sido relacionados.

Así pues, se trata de comparar una solicitud del usuario con un conjunto de productos para recomendarle al usuario los productos que más le puedan interesar. Básicamente, los sistemas recomendadores se pueden enfocar de 2 maneras diferentes según de dónde se obtiene la información para realizar la recomendación. Si las características para la comparación se obtienen del producto en cuestión, se dice que es una recomendación basada en contenido. Por ejemplo si para recomendar libros, utilizas características como el título, los autores, el género literario, etc, entonces estarías haciendo una recomendación basada en contenido. El otro enfoque es la aproximación por filtrado colaborativo, donde las características para la recomendación provienen del entorno social del usuario. De esta manera, para el ejemplo del recomendador de libros, si se tiene un perfil del usuario con su edad, profesión, géneros favoritos, últimas compras, perfiles de usuarios similares… y se utiliza dicha información para la recomendación, entonces se trataría de un recomendador colaborativo. Muchas veces los sistemas recomendadores combinan ambas aproximaciones para obtener mejores resultados.

Existen sistemas recomendadores para multitud de dominios de aplicación: libros, películas, viajes, ropa, música, páginas web… A continuación expondremos algunos de estos recomendadores.

Por ejemplo, el recomendador de gafas iCare. En este sistema creas tu perfil de usuario, donde es imprescindible añadir una foto para que el sistema pueda mostrarte una imagen donde te veas con las gafas que hayas seleccionado. El usuario crea su perfil y solicita al sistema que le proponga una gafas, el sistema analiza los rasgos de la cara del usuario y busca en su base de conocimiento (donde tiene todos los modelos de gafas disponibles) para finalmente recomendarle las gafas que mejor se ajustan a sus características. Una detalle interesante de este recomendador es la posibilidad de ver tu foto con las gafas puestas para poder decidir tú mismo si las gafas te sientan bien o no. En la *figura 2.5* que se muestra a continuación, se puede observar que al usuario registrado se le recomiendan 4 modelos de gafas (imagen de la derecha) y que seleccionando uno de los modelos de gafas recomendados se puede ver una descripción con el precio, material, tamaño de las gafas... (imagen central). Por último, al usuario se le ofrecen 2 posibilidades: comprar las gafas (Buy) o seguir buscando unas gafas parecidas a las que ha seleccionado (More Like This). Se trata pues de un recomendador conversacional como explicaremos más adelante.

Figura 2.5 -- Recomendador de gafas iCare

Otro sistema recomendador muy popular en Internet es Amazon (http://www.amazon.com). En esta página se pueden buscar libros, películas, productos electrónicos, menaje del hogar… Para todas estas áreas la aplicación propone inicialmente una lista de productos de oferta, últimas novedades, los más vendidos, etc. A partir de lo que el usuario seleccione de esa lista inicial, el sistema utiliza su sistema de recomendación para guiar al usuario hacia el producto que esté buscando. Esta técnica utilizada para obtener la consulta inicial es bastante efectiva porque no necesita mucha información del usuario para empezar a recomendar (el usuario se cansa rápidamente si tiene que rellenar muchos formularios hasta obtener resultados). Amazon también permite que los usuarios se registren rellenando un formulario con sus preferencias. De esta manera, el proceso de recomendación suele ser más exitoso ya que el sistema tiene más información de lo que el usuario está buscando. Además, al registrarte el sistema te manda periódicamente boletines personalizados al correo electrónico con las últimas novedades.

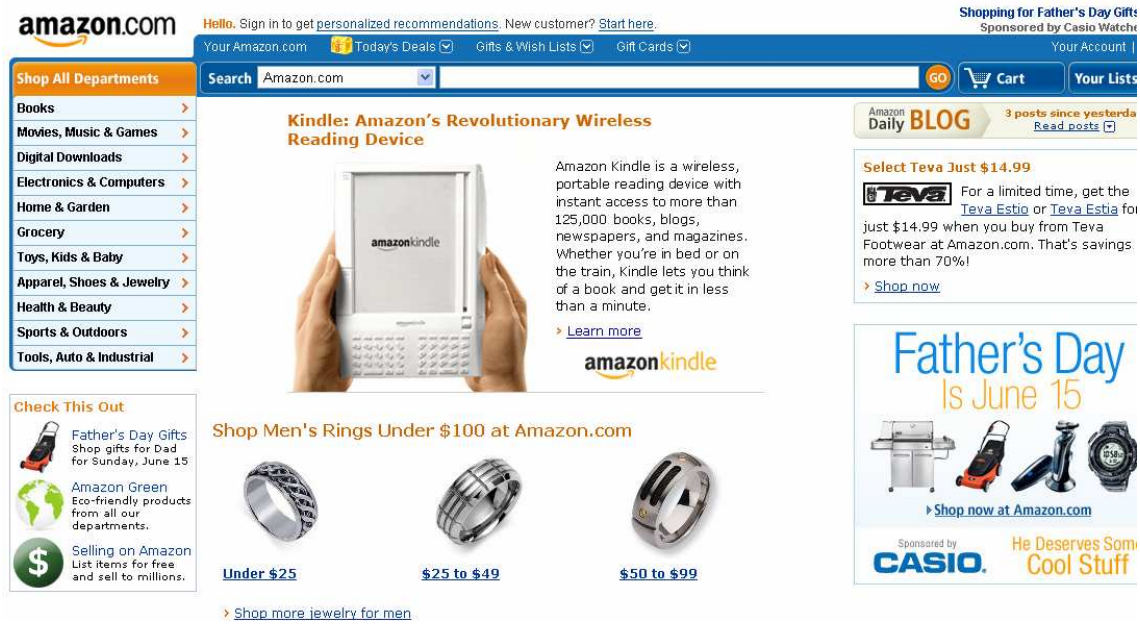


Figura 2.6 -- Recomendador Amazon

Otro sistema recomendador en Internet es Meetic (http://www.meetic.es), se trata de un recomendador de parejas. El usuario debe registrarse rellenando varios formularios donde se especifica tu perfil (edad, sexo, profesión, lugar de residencia, inquietudes, ocio…) y el perfil de la pareja que estás buscando. El sistema pone en contacto al usuario, a través de correo electrónico, con personas que tienen perfiles similares al suyo y que buscan lo mismo. Este tipo de recomendadores tienen bastante éxito ya que puedes contactar con personas con gustos similares a los tuyos de manera rápida y sencilla.

Figura 2.7 -- Recomendador de parejas Meetic

Una vez vistos algunos ejemplos de sistemas recomendadores, pasaremos a centrarnos en nuestro recomendador de artículos de la biblioteca de la UCM y en las técnicas de razonamiento empleadas. Tanto para los recomendadores basados en contenido como para los recomendadores basados en perfil de usuario, una técnica muy utilizada para realizar la similitud entre productos, es el razonamiento basado en casos (CBR). Utilizan el ciclo CBR para recomendar un producto de la base de casos según la consulta realizada por el usuario. Es interesante resaltar la idea de recomendador en contraposición a la de buscador. Un buscador realiza una recuperación exacta, es decir, devuelve casos total o parcialmente iguales a la consulta realizada. En cambio, un recomendador tiene la capacidad de ofrecer productos que no tienen porque parecerse exactamente a la consulta, sino que mediante su razonamiento recupera casos que son similares a la consulta aunque no tengan los campos iguales a los que pidió el usuario. Por eso, se dice que un recomendador realiza una recuperación aproximada. Por ejemplo, en nuestro proyecto trabajamos con artículos de revistas de la biblioteca de la UCM, si el usuario solicita artículos de cierta materia y de un determinado autor, el recomendador le ofrecerá artículos no solo de esa materia y autor, sino también artículos de autores y materias similares. Así pues, la capacidad de inferir casos similares dependerá de la calidad de las técnicas de similitud con las que se dote al sistema recomendador.

Realizar un sistema recomendador desde cero es una tarea compleja. El desarrollo de una aplicación de este tipo sería más sencillo si se pudiesen reutilizar componentes genéricos ya implementados. jCOLIBRI2 ofrece una librería de plantillas de recomendadores [8] que realizan esta función. Así, el diseñador solo debe escoger la plantilla mas parecida al sistema que desea implementar y adaptarla a sus necesidades. De esta manera, el diseñador se ahorra el diseño de partes que son comunes a cualquier recomendador y se puede centrar en funciones específicas de su sistema. A continuación explicaremos como son estas plantillas y como utilizarlas para desarrollar un sistema recomendador.

La primera distinción entre sistemas recomendadores se basa en la interacción que tiene el usuario con el sistema. Con esta idea, distinguimos entre recomendadores de una sola interacción con el usuario (Single-Shot Systems – *Figura 2.8*) y recomendadores de varias interacciones con el usuario (Conversational Systems – *Figura 2.9*). Los primeros reciben una consulta del usuario, devuelven sus recomendaciones y termina. Los conversacionales por el contrario, después de recuperar sus recomendaciones permite al usuario volver a redefinir su consulta en función de los casos recuperados, así se va refinando más la consulta y los resultados previsiblemente serán mejores. Estas interacciones se pueden repetir tantas veces como el usuario desee.

Estos recomendadores a su vez se dividen en varias tareas y subtareas como muestran las siguientes figuras. Como se puede observar hay varias tareas comunes a los dos tipos de recomendadores (One-Off Preference Elicitation, Retrieval y Display Item List) y otras que son propias de los recomendadores conversacionales (Iterated Preference Elicitation). Cada una de estas tareas se puede realizar de diversas maneras y es lo que pasaremos a explicar a continuación.

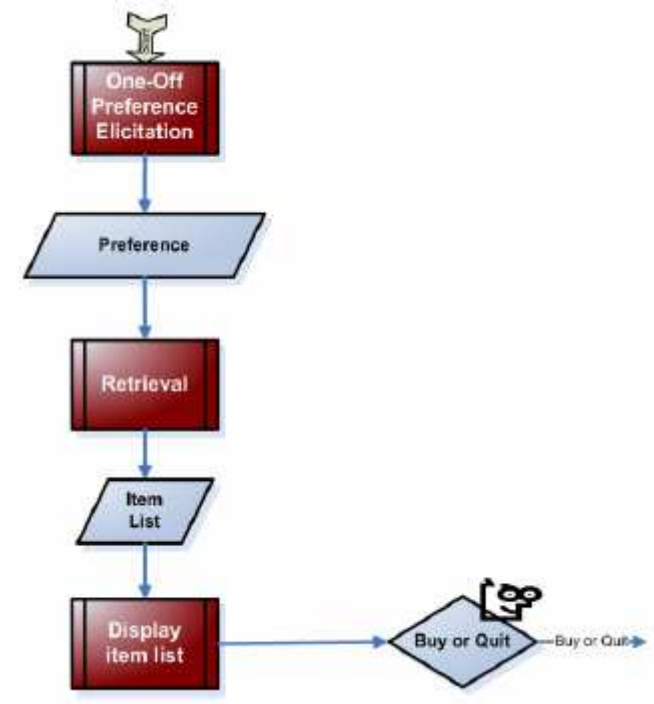


Figura 2.8 -- Recomendador Single-Shot

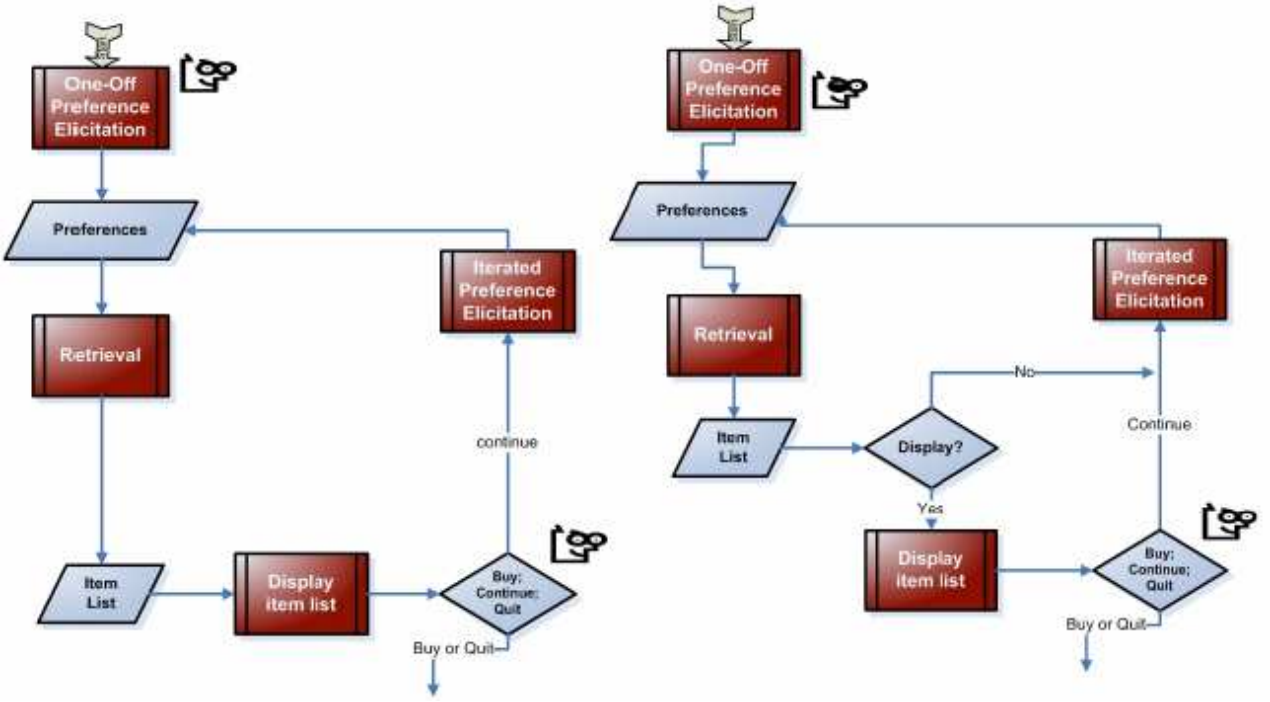


Figura 2.9 -- Recomendadores Conversacionales

- **One-Off Preference Elicitation**

Esta tarea tiene como objetivo obtener la consulta inicial del usuario. jCOLIBRI2 ofrece tres posibilidades para realizar dicha tarea. La primera posibilidad es obtener la consulta a través de un perfil de usuario (Profile Identification). Para ello el usuario debe identificarse y el sistema obtiene su perfil de una base de datos de usuarios registrados. Si es un usuario nuevo el sistema debe proporcionar un servicio para registrarse donde se incluyan sus gustos, preferencias y datos personales que sean relevantes para el recomendador. El perfil puede contener las preferencias del usuario, descripciones de productos seleccionados anteriormente, descripciones de las últimas sesiones del usuario con el sistema y/o valoraciones de productos hechas por el usuario.

Una alternativa a este método basado en perfil es obtener la consulta inicial con información que proporciona directamente el usuario sobre el producto que está buscando (Initial Query Elicitation). Esta tarea se puede realizar de varias formas. Una posibilidad es rellenando un formulario con diversos campos (Form Filling). Otra manera es mediante preguntas que se realizan al usuario y en base a sus respuestas se obtiene la consulta inicial (Navigation by Asking). También se puede proponer al usuario una lista de productos y que escoja el más similar a lo que está buscando (Navigation by Proposing). De esta manera, con la descripción del producto seleccionado junto a una opinión del usuario sobre el producto se obtiene la consulta inicial.

Ya que las dos posibilidades anteriores (con perfil de usuario o preguntando al usuario) pueden ser complementarias, la tercera posibilidad para obtener la consulta inicial es combinar dichas opciones (Profile Identification & Query Elicitation).

- **Retrieval**

La tarea de recuperación consiste en extraer de la base de casos del recomendador los casos más similares a la consulta inicial. Es una fase común para cualquier tipo de recomendador y de cualquier sistema CBR como se ha explicado en apartados anteriores. La recuperación es una tarea compleja con muchas posibles implementaciones y que depende de la técnica empleada en la tarea anterior de obtención de la consulta (One-Off Preferente Elicitation), ya que si por ejemplo se empleó la técnica de perfil de usuario, entonces tendrá que utilizar otros perfiles de usuarios similares para la recuperación. Se puede realizar la recuperación usando filtros que desestimen los casos que no cumplan ciertas condiciones, o buscando un conjunto de casos que sean representativos de toda la base de casos. En cualquier caso, además de una fase de selección de casos relevantes se debe realizar una puntuación de cada caso, para así obtener los más similares. Esto es lo que hace el método k-NN (k-Nearest Neighbours) que es el que utilizamos en nuestro sistema recomendador.

- **Display item list**

Esta tarea simplemente muestra la lista de casos recuperados. La forma en que se muestran al usuario suele ser mediante una tabla con los atributos más relevantes, para que el usuario escoja el producto que desea o sobre el que quiere seguir realizando el proceso de recuperación (en el caso de un recomendador conversacional).

- **Iterated Preference Elicitation**

Esta tarea es propia de los sistemas recomendadores conversacionales. Consiste en volver a interactuar con el usuario de manera que pueda refinar su consulta inicial. El sistema ya ha propuesto un conjunto de productos y ahora el usuario tiene más información sobre lo que hay, y puede refinar la consulta inicial combinándola con una crítica sobre los casos recuperados. Por ejemplo, el sistema le ofrece una serie de productos y el usuario puede escoger uno y pedirle un caso "como este pero más barato" o "como este pero más cerca" si se trata de recomendar viajes de vacaciones. De esta manera el sistema se realimenta con información que le ofrece el usuario.

Al igual que la tarea One-Off Preference Elicitation, jCOLIBRI2 propone tres posibilidades: mediante Form-Filling donde el usuario rellena un formulario, con Navigation-by-Asking realizando preguntas al usuario (el sistema debe usar heurísticas para escoger la mejor pregunta en función de respuestas anteriores) o con Navigation-by-Proposing donde se muestran unos productos y se pide al usuario un comentario o crítica (del tipo "como este pero…") que guie al sistema hacia el producto que se desea encontrar.

3. PROCESO CATALOGADOR

3.1 Adquisición de Conocimiento

En un principio, para adquirir conocimiento sobre el proceso de catalogación de artículos en una biblioteca, realizamos varias reuniones con una pseudo-experta en catalogación (Laura Henche – Programadora del Dpto. de Automatización de la Biblioteca) que nunca había participado en la catalogación de un documento pero que tenía muchos conocimientos sobre el tema, ya que ha estado trabajando con expertos en catalogación. En una biblioteca se catalogan diferentes tipos de documentos, pero tomamos la decisión de centrarnos para empezar nuestro trabajo en la catalogación de artículos del portal de revistas electrónicas. Ella nos inicio en los conceptos necesarios para catalogar un artículo, en los metadatos que se deben extraer de los artículos durante el proceso de catalogación y nos proporcionó información acerca de las estructuras de datos que se utilizan en la biblioteca para organizar los artículos.

Tras investigar sobre los distintos metadatos que se podían extraer de un artículo, qué representaba cada uno de ellos, y la manera en la que se obtenían (si se cogían de la revista en la que se publica el artículo o del mismo artículo), realizamos una reunión con una experta en catalogación que actualmente es la Subdirectora de la Biblioteca de la Facultad de Filosofía de la UCM, el objetivo de esta reunión era comprender perfectamente el significado de cada uno de los metadatos del artículo, y aprender los pasos que se siguen en la catalogación de un artículo para ver cómo podíamos enfocar nuestra aplicación. Los temas que queríamos abordar con la experta eran:

- Etapas en el proceso de catalogación de un artículo (Entradas y Salidas)
- Explicación de cada uno de los metadatos del artículo
- Profundizar en la extracción del metadato *Materia 2* ya que nos parecía un metadato muy interesante para tratarlo en nuestra aplicación.

Ella nos explicó paso a paso cómo se cataloga un artículo en la biblioteca:

- Inicialmente se recibe como entrada un artículo en PDF, la revista de la que proviene y la ruta en la que se encuentra el artículo.

- Lo primero es identificar la revista de la que viene el artículo, si es una revista nueva hay que extraer los datos relativos a la revista, e incluir la nueva revista en las bases de datos.

- Para añadir el nuevo artículo se indica el directorio en el que se encuentra y se añaden un conjunto de metadatos que provienen de la revista:

    - Volumen
    - Número
    - Año de publicación
    - Área (Humanidades, Ciencias Sociales, Ciencias de la Salud, Ciencias)

- Embargado : indica si el artículo está o no accesible, depende de la fecha de publicación ya que "embargo = (fecha de publicación del fascículo + meses embargados de la revista) > fecha actual "

- Una vez identificada la revista se busca la página inicial y la página final del artículo para formar el nombre que se le va a asignar, ya que los nombres asignados a los artículos siguen una estructura especial, por ejemplo:

REFA93931301 20A

REFA	Posición **1,4**	Código de la Revista
93	Posición **5,2**	Año de inicio del Volumen
93	Posición **7,2**	Año de terminación del Volumen
1	Posición **9,1**	Número de Volumen de ese año
3	Posición **10,1**	Número total de Volúmenes de ese año.
0120	Posición **11,4**	Número de página donde comienza y termina el articulo relleno con ceros a la izquierda.
A	Posición **15,1**	Diferenciador de código PDF Valor por defecto: **A**

- El siguiente paso es obtener los metadatos propios del artículo, a parte de los que ya se han extraído de la revista tenemos:

  - Clase: Identifica el tipo de documento (portada, artículo, editorial, noticia…), en algunos casos viene indicada en el mismo artículo y otras veces se puede ver en el sumario de la revista donde se tiene la estructura que sigue la misma. Si no viene especificado en ningún sitio debe extraerlo el bibliotecario haciendo uso de su experiencia.

  - Página de inicio y Página de fin: Se extraen del nombre que se le ha asignado al artículo, como ya hemos explicado anteriormente.

  - Título y Autor: Se obtienen del artículo en cuestión. Cada artículo puede tener uno o varios autores. El título es único e indispensable para incluir el artículo en el portal de revistas electrónicas.

  - Resumen y Palabras Clave: Se extraen del artículo si vienen y si no aparecen entonces el bibliotecario los puede obtener a partir de la lectura del artículo.

  - Materia 1: Es la materia principal de la que trata el artículo, suele coincidir con la *Materia 1* que tiene asignada la revista a la que pertenece el artículo, pero puede ser diferente.

  - Materia 2: Son submaterias más específicas dentro de la *Materia 1* (a cada *Materia 2* sólo le corresponde una *Materia 1*) y se puede asignar más de una a cada artículo. El bibliotecario obtiene este campo a partir del resumen y las palabras clave si existen en el artículo, si no es así debe realizar una lectura más profunda del artículo

para extraer este metadato. Existen varias *Materias 2* de valor "General", sólo que cada una se corresponde con una *Materia 1* determinada.
Podríamos tener por ejemplo como *Materia 1* de un artículo "Informática" y como *Materia 2* ya más específica "Internet".

Una vez se han extraído todos los metadatos necesarios para la catalogación, mediante la aplicación que utilizan en la biblioteca se guarda el artículo con todos sus metadatos, de forma que queda añadido al portal de revistas electrónicas.

Tras recopilar toda la información necesaria para entender el proceso de catalogación de artículos, llegamos a la conclusión de que era viable la elaboración de una aplicación que realizara una parte del proceso de catalogación de manera automática, puesto que disponemos de las herramientas necesarias para el tratamiento de textos y por lo tanto, para la extracción de algunos de los metadatos.

Una vez obtenida toda esta información y estudiada la viabilidad del desarrollo de la aplicación dimos por finalizada la fase de adquisición del conocimiento, para dejar paso a la fase de desarrollo.

3.2 Desarrollo

Una vez adquirido todo el conocimiento necesario para la catalogación de un artículo del portal de revistas electrónicas, revisamos a fondo toda la información recibida y estudiamos de qué manera podíamos automatizar la extracción de cada uno de los metadatos del artículo. Debido a que no todos los artículos seguían la misma estructura se hacía un poco difícil la extracción automática de algunos metadatos como por ejemplo el *Título* y el *Autor,* que de manera manual son más fáciles de obtener. Por ello, llegamos a la conclusión de centrarnos en una primera fase dentro del proceso de catalogación en la que extraeríamos un conjunto de metadatos:

- Resumen y Abstract
- Palabras Clave y Keywords

} Siempre que estén incluidos en el artículo

- Materia 2
- Materia 1

Esta primera fase debería ser complementada con una revisión por parte del bibliotecario de los metadatos extraídos automáticamente, seguidas de otra fase donde se extraen el resto de metadatos, tras la cual se añadiría el artículo ya catalogado al portal de revistas electrónicas.

Decidimos extraer automáticamente los metadatos *Resumen, Abstract, Palabras Clave y Keywords* ya que siguen un patrón claramente diferenciable y podemos utilizar expresiones regulares para extraerlos. Si el artículo no contiene Resumen, Palabras Clave, Abstract o Keywords, los campos extraídos serán vacíos y será labor del bibliotecario rellenarlos.

En cuanto al metadato *Materia 2* requiere de la experiencia del bibliotecario, ya que es uno de los campos más complejos de extraer, por eso decidimos profundizar más en la extracción automática de este metadato y así facilitar el proceso de catalogación. La manera de obtenerlo fue aplicando técnicas basadas en conocimiento (CBR textual). Decidimos que teniendo una base de casos que contenga los artículos ya catalogados previamente, podríamos realizar una similitud del

artículo a catalogar con todos los artículos incluidos en la base de conocimiento, extrayendo de la base de conocimiento los artículos más parecidos y así poder asignar al artículo la *Materia 2* de los artículos más parecidos.

Como base de conocimiento tenemos una selección representativa de artículos ya catalogados, esta base de conocimiento es fácilmente extensible con los nuevos artículos que se van catalogando de forma que se va aumentando la diversidad de artículos y se van cubriendo más casos. Para que esta base de conocimiento no crezca demasiado habría que aplicarle un mantenimiento para que se quede siempre con una selección representativa de artículos.

Respecto al metadato *Materia 1,* como a cada *Materia 2* sólo le corresponde una *Materia 1*, una vez que obtengamos la *Materia 2*, obtenemos la *Materia 1* correspondiente.

Por lo tanto, nuestra aplicación recibiría como entrada el artículo a catalogar en PDF y devolvería como salida los metadatos anteriormente mencionados. Como restricción de entrada para nuestra aplicación son que los artículos no estén protegidos.

Esta aplicación sería fácilmente extensible a otro tipo de recursos existentes en la biblioteca, como por ejemplo las Tesis, E-Prints… puesto que el proceso sería el mismo, lo único que cambiaría serían los metadatos propios de cada recurso.

Ahora en las dos subsecciones siguientes vamos más en detalle los pasos que seguimos en el desarrollo.

3.2.1 Extracción de los metadatos

Para poder extraer todos estos metadatos, primero hay que transformar los artículos que recibimos en '.pdf' a textos en '.txt'. Tuvimos una gran dificultad para encontrar la manera de transformar los textos a '.txt', hasta que nos facilitaron la aplicación que se utiliza en la biblioteca. Con ella conseguimos transformar todos los '.pdf' a '.txt' manteniendo el formato y sin problemas, excepto con los artículos protegidos que no se podían transformar a '.txt' y con los artículos '.pdf' que están escaneados que los transforma de manera errónea a '.txt' ya que interpreta mal los caracteres. Por ello, una de las restricciones de entrada a nuestra aplicación es que los artículos a catalogar no estén protegidos y no estén escaneados.

RESUMEN, PALABRAS CLAVE, ABSTRACT Y KEYWORDS

Para la extracción de estos metadatos utilizamos expresiones regulares. Vimos que se suele cumplir en la mayoría de los artículos que aparece el *Resumen* seguido de las *Palabras Clave* que normalmente ocupan una línea, y tras ellas aparece el *Abstract* seguido de las *Keywords.*

Siguiendo esta secuencia de aparición las expresiones regulares que construimos para extraer cada uno de estos campos seguirían estas reglas:

- Resumen: Pensamos en extraer todo el texto comprendido entre la palabra Resumen y las Palabras Clave, pero como puede ocurrir que un artículo tenga Resumen pero no tenga Palabras Clave y pase directamente al Abstract o a la introducción, al final pusimos que extraiga el texto hasta las Palabras Clave o el Abstract. Al hacerlo así nos encontramos con el problema de que por la forma en la que se ejecutan las expresiones regulares, si un artículo contenía el Resumen seguido de las Palabras Clave seguidas del Abstract, el texto

que nos extraía como Resumen contenía también las Palabras Clave. La única solución que encontramos fue procesar el Resumen una vez extraído y quitarle las Palabras Clave si las tenía.

- Palabras Clave: En este caso, como las palabras clave aparecen en una única línea lo que hacemos es extraer la línea que sigue a la expresión Palabras Clave.

- Abstract: Su extracción es similar a la del Resumen, sólo que en este caso se extrae el texto comprendido entre la palabra Abstract y las Keywords o la introducción.

- Keywords: Su extracción es similar a la de las Palabras Clave, sólo que en este caso se extrae la línea que sigue a la palabra Keywords.

Como ya hemos visto hay artículos que puede que no contengan alguno de estos campos, en ese caso el campo que no aparece en el artículo queda vacío. En algunos casos debido a la estructura que sigue el artículo no extrae bien estos campos ya que los mezcla, como ocurre por ejemplo si salen en un lateral las *Palabras Clave* y en otro lateral el *Resumen*,

MATERIA 2

La extracción de este metadato es algo más compleja, y como ya hemos mencionado anteriormente vamos a utilizar técnicas basadas en conocimiento (CBR Textual) para obtener la similitud entre el artículo a catalogar y la base de conocimiento. Las pruebas que hicimos inicialmente fueron con un conjunto reducido de artículos de los cuales conocíamos la *Materia 2* que tenían asignados. En todas las pruebas realizadas seleccionábamos un artículo al azar de la base de conocimiento, lo eliminábamos de ella y lo utilizábamos como query (artículo a catalogar).

Empezamos haciendo pruebas con OPENNLP. Primero probamos comparando los textos completos para hallar la similitud, pero era muy ineficiente, ya que para una prueba con diez artículos incluidos en la base de conocimiento tardaba alrededor de los cuatro minutos, por lo que para todo el conjunto de artículos que íbamos a tener que utilizar en nuestra aplicación este método se hacía intratable. Para reducir los tiempos pensamos en comparar los artículos mediante el *Resumen* y las *Palabras Clave,* pero como no todos los artículos tienen estos campos nos dimos cuenta que no podíamos utilizar este método para calcular la similitud.

Una solución que se nos ocurrió fue utilizar sólo las 50 primeras líneas de cada artículo para realizar la similitud, pensamos en esta cifra ya que así se incluye la introducción del artículo y el *Resumen* y las *Palabras Clave* de aquellos artículos que los tuvieran, que creemos que es la parte más relevante del documento. De esta forma, dejamos los artículos almacenados en la base de conocimiento sólo con sus 50 primeras líneas, y para cada artículo a catalogar extraemos también sus 50 primeras líneas. Con este método se mejoro mucho el tiempo de ejecución, y los resultados obtenidos eran similares.

Tras estas primeras pruebas con OPENNLP, probamos a calcular la similitud mediante LUCENE. Siguiendo con la idea de las 50 primeras líneas, mediante LUCENE conseguimos mejores tiempos de ejecución y aumentar el porcentaje de aciertos, por ello tomamos la decisión de utilizar LUCENE para hallar la similitud entre artículos y nos centramos en realizar pruebas con este método con una mayor cantidad de artículos en la base de conocimiento. Durante la realización de estas pruebas, nos dimos cuenta que si ampliábamos a extraer las 100 primeras líneas en los textos que comparábamos se mejoraban los resultados de similitud. Al aplicar

LUCENE para hallar la similitud entre el artículo a catalogar y los artículos pertenecientes a la base de conocimiento, se obtiene para cada artículo de la base de conocimiento un grado de similitud cuyo valor está comprendido entre 0 y 1, y de esos artículos seleccionamos los "k" artículos más parecidos, siendo k una variable que hemos fijado a 3 como veremos más adelante.

LUCENE utiliza una lista de palabras vacías para almacenar aquellas palabras que no tiene en cuenta a la hora de realizar la similitud porque no tienen ninguna relevancia. Estas palabras están tanto en inglés como en español y son preposiciones, artículos, determinantes…Nosotros además de estas palabras, hemos añadido en la lista de palabras vacías otras palabras que no tienen ninguna importancia a la hora de realizar la similitud ya que son muy frecuentes en todos los artículos, como por ejemplo: resumen, autor, introducción, abstract…

Un problema que nos encontramos con LUCENE fue que el artículo que se quería catalogar no podía contener algunos caracteres, porque si los tenía el algoritmo fallaba y no funcionaba correctamente. Lo que tuvimos que hacer para solucionar este problema fue procesar el texto del artículo a catalogar antes de aplicarle LUCENE, este procesamiento consistía en eliminar esos caracteres tales como @, $, *, /, ", ', +, ^… La dificultad que tuvimos fue encontrar todos estos caracteres que hacían que el algoritmo fallase, ya que hubo que realizar muchas pruebas para encontrar todos.

Vimos que si al artículo le asignábamos sólo la *Materia 2* del artículo más similar los resultados no siempre eran correctos, y que muchas veces no era mucha la diferencia del grado de similitud entre los tres artículos más parecidos y el artículo a catalogar, además se daba el caso muchas veces de que el *Materia 2* del segundo artículo más parecido o del tercero coincidía con el del artículo que queríamos catalogar, normalmente con un grado de similitud mayor a 0.5. Todo esto unido al hecho de que durante el periodo de Adquisición de Conocimiento vimos que a un artículo se le puede asignar más de una *Materia 2,* nos hizo tomar la decisión de asignar al artículo que se está catalogando la *Materia 2* del los 3 artículos de la base de conocimiento que recupera LUCENE que tienen un grado de similitud mayor a 0.5.

Con esto conseguimos aumentar el porcentaje de aciertos al 70% como se verá más adelante en las pruebas masivas que realizamos usando técnicas de validación cruzada. Hay que tener en cuenta que la base de conocimiento que estábamos utilizando en estas pruebas, era un conjunto de artículos seleccionado por la pseudo-experta en catalogación Laura Henche que intentó incluir en el conjunto artículos de todas las revistas y de varios idiomas para formar un conjunto significativo, y por lo tanto no cubría todas las materias 2 existentes, por lo que nos quedaba la labor de intentar aumentar este porcentaje de aciertos realizando una buena selección de artículos para la base de conocimiento.

MATERIA 1

Su extracción es prácticamente automática una vez que se le ha asignado la *Materia 2* al artículo que se está catalogando. Lo único que hay que hacer es asignarle la *Materia1* correspondiente a la *Materia 2* asignada, ya que como vimos en el periodo de Adquisición de Conocimiento, a cada *Materia 2* sólo le corresponde una *Materia 1.*

De esta forma se extraen todos los metadatos que vamos a mostrar como solución del artículo que se quiere catalogar. En resumen nuestra aplicación trabajaría con los siguientes datos:

CASOS: Artículos ya catalogados que tenemos en la base de conocimiento

- Descripción:
  - Nombre del PDF
  - Texto: Primeras 100 líneas del artículo
- Solución:
  - Materia2
  - Materia1

QUERY: Artículo que se va a catalogar

- Texto: Primeras 100 líneas del artículo

RESULTADO: Metadatos extraídos como solución de la catalogación

- Resumen
- Abstract
- Palabras Clave
- Keywords
- Materia2
- Materia1

Los artículos de la base de conocimiento han sido previamente procesados para convertirlos de ".PDF" a ".txt" y se han almacenado en un directorio para su posterior uso. Laura Henche nos proporcionó un conjunto de tablas en SQL de las bases de datos de la Biblioteca. Estas tablas eran:

- Clasifica (Id, Nombre, IdMateria2): formada por el nombre de los artículos y el código numérico de su *Materia 2* asignada.
- Materias 2 (IdMateria2, Materia2): formada por el código numérico de cada *Materia 2* y su nombre correspondiente.
- Materias 1 (IdMateria1, Materia1): formada por el código numérico de cada *Materia 1* y su nombre correspondiente.
- Engloba (IdMateria1, IdMateria2): relaciona cada *Materia 2* con su *Materia 1* correspondiente.

Nosotros creamos una vista para relacionar estas tablas. A partir de esta vista se realiza un mapeo para cargar en memoria los casos de la base de conocimiento. Los casos están formados por una "descripción" y una "solución", en la "descripción" se mapea el nombre del artículo y se cargan las 100 primeras líneas del texto del artículo, este campo se obtiene a partir del archivo ".txt" generado previamente. En la "solución" se mapean el *Materia 2* y el *Materia 1,* tanto en código numérico como en nombre. Para el mapeo de todos estos atributos de la base de conocimiento se utiliza el conector de base de datos (DataBaseConnector) que contiene el framework jCOLIBRI2.

Al trabajar con las bases de datos nos encontramos algunos problemas. Tuvimos que procesar todas las tablas SQL que nos proporcionó Laura Henche para adaptarlas a la configuración de nuestro conector ya que la sintaxis era algo distinta. Además, al principio debido a la codificación de las tablas y a un error al utilizar una librería incorrecta, no se podían cargar datos que contuvieran acentos o ñ's.

Nuestra aproximación usando CBR hace que los resultados y la eficiencia de la aplicación dependan directamente de la cantidad y calidad de los casos almacenados. Si usamos una base de casos muy grande se verá degradado el rendimiento pero los resultados en términos de precisión y recall serán muy buenos. Al contrario, una base de casos pequeña mejorará la eficiencia a costa de la calidad.

El objetivo es conseguir una base de casos representativa, con un conjunto de casos que proporcione suficiente cobertura para las posibles consultas, sin degradar ni la eficiencia, ni la calidad del resultado. En el siguiente apartado vamos a ver cómo obtener una base de casos representativa.

3.2.2 Aplicación del mantenimiento

Como no era viable comparar cada artículo a catalogar con todos los artículos que hay en la biblioteca que están ya catalogados y que tienen asignada una *Materia 2* (alrededor de 3500 artículos), necesitábamos hacer una selección representativa de artículos para formar la base de conocimiento. Esta selección la realizamos aplicando algoritmos de mantenimiento (RENNNoiseReduction y ICFRedundancyRemoval) incluidos en el framework jCOLIBRI2.

El primer algoritmo (RENNNoiseReduction) elimina el ruido de la base de conocimiento, esto significa que elimina los artículos de la base de conocimiento, tales que al aplicarles la función de similitud se obtiene como artículos más parecidos artículos con diferente *Materia 1* al artículo en cuestión. Por ejemplo, podríamos tener un artículo con *Materia 1* "Biología" y al realizar la similitud con el resto de artículos de la base de conocimiento sus tres artículos más parecidos tienen como *Materia 1* "Ciencias de la Información", entonces el algoritmo borraría este artículo de la base de conocimiento ya que no es un buen caso representativo.

El segundo algoritmo (ICFRedundancyRemoval) elimina los artículos redundantes de la base de conocimiento, ya que si por ejemplo tenemos dos artículos con *Materia 2* "Botánica" y los dos son muy similares, no aporta nada tener los dos artículos incluidos en la base de conocimiento ya que sólo con uno de ellos sería suficiente.

De esta forma, al aplicar el mantenimiento sobre la base de conocimiento formada por aproximadamente 3500 artículos queda reducida a un conjunto representativo de aproximadamente 450 artículos, que es una cantidad tratable para nuestra aplicación.

Nuestra aplicación irá introduciendo en la base de conocimiento los artículos que hemos considerado que han sido correctamente catalogados, esto quiere decir que sus artículos más similares tienen todos la misma *Materia 1*. Una vez la base de conocimiento supere el umbral de los 800 artículos se vuelve aplicar el mantenimiento para reducir de nuevo el conjunto representativo de artículos que forman la base de conocimiento. La finalidad de ir introduciendo nuevos artículos catalogados en la base de conocimiento es que la aplicación "aprenda", es decir, que contenga una mayor diversidad de casos.

3.2.3 Interfaz de la aplicación

Hemos realizado una interfaz sencilla. Al principio sólo permitía la catalogación de artículos de uno en uno, y para cada artículo catalogado mostraba su solución (Nombre, IdMateria2, IdMateria1, Resumen, Palabras Clave, Abstract y Keywords). Posteriormente, para la

realización de pruebas masivas y pensando que sería más cómodo poder catalogar más de un artículo a la vez, modificamos la interfaz para permitir que el usuario pueda seleccionar un directorio en el que se encuentran todos los artículos a catalogar. Se catalogan todos los artículos que se encuentran en el directorio seleccionado y luego se permite que el usuario vaya viendo la solución de cada artículo catalogado mediante los botones "anterior" y "siguiente". El usuario puede pulsar sobre el botón "Catalogar más artículos" para volver a seleccionar otro directorio que contenga nuevos artículos a catalogar.

Un ejemplo de la aplicación en donde se muestra la solución de un artículo después de su catalogación sería:

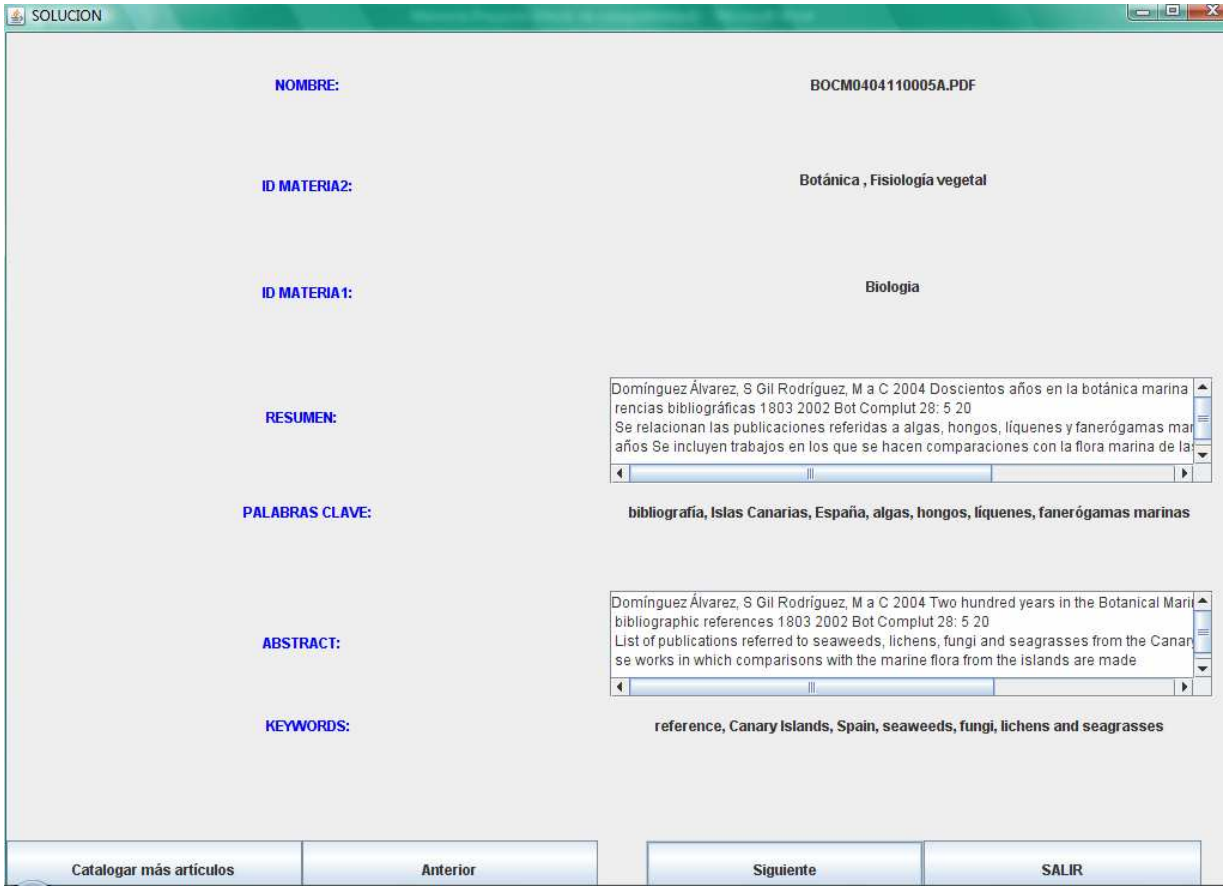


Figura 3.1

Esta interfaz la diseñamos como ejemplo para el prototipo que se presentó en la Biblioteca, pero no es la interfaz definitiva, puesto que era necesaria la opinión de la Biblioteca. Una vez la valoraron, nos presentaron una serie de nuevos requisitos para el prototipo final de los que hablaremos en el apartado de Integración con la Biblioteca.

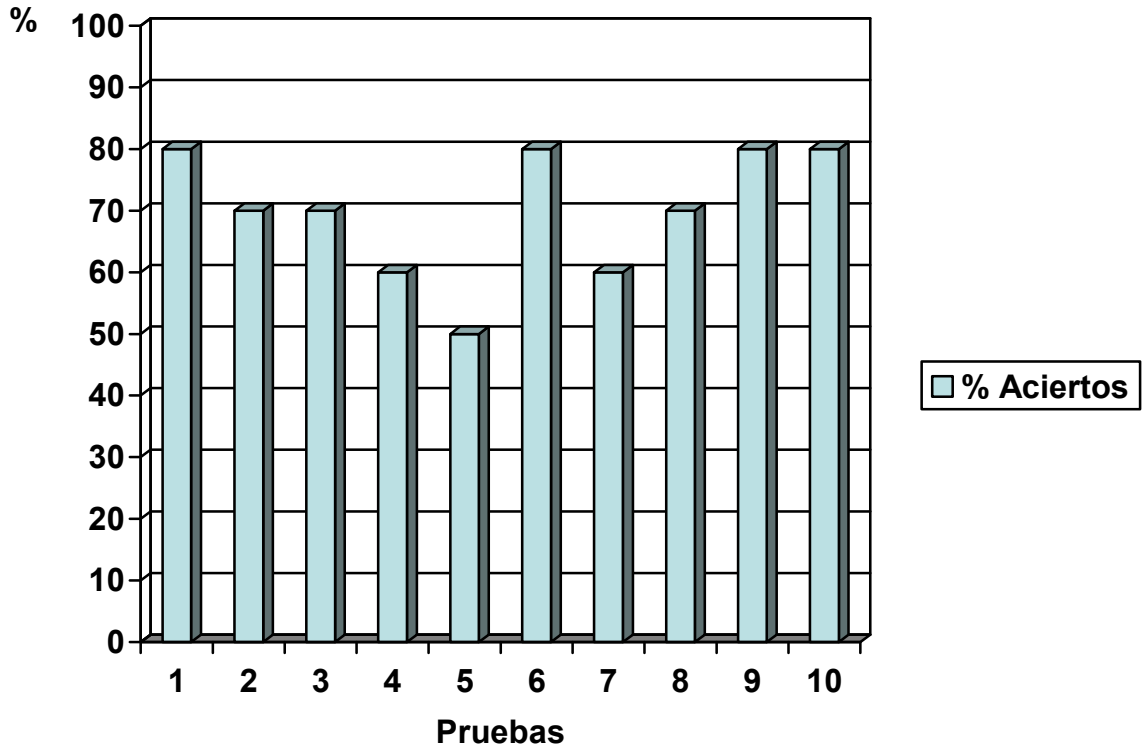
3.3 Experimentos

Para la realización de pruebas masivas aplicamos la técnica del N-Cross Validation, esta técnica consiste en seleccionar aleatoriamente N artículos para catalogar de los cuales se sabe su

solución (la *Materia 2* y la *Materia 1* que tienen asignada), estos artículos hay que borrarlos de la base de conocimiento para realizar su catalogación. Se apunta el porcentaje de aciertos y se repiten las pruebas con otra selección de N artículos diferentes, y así sucesivamente hasta realizar N iteraciones.

Nosotros fijamos el valor de N a 10, por lo que realizamos 10 tandas de pruebas catalogando 10 artículos en cada iteración.

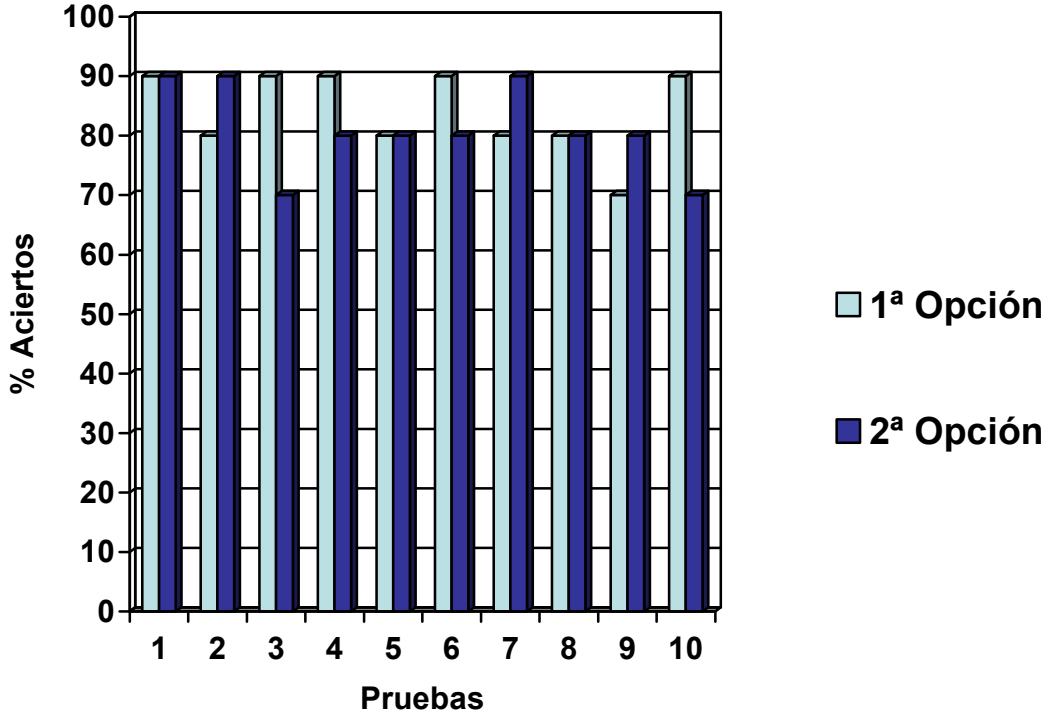
La primera prueba masiva la realizamos con una base de conocimiento formada por 512 artículos seleccionados por la pseudo-experta Laura Henche, y que como hemos mencionado anteriormente no cubrían todas las Materias 2 posibles. Aún así los resultados obtenidos fueron bastante buenos:



RESULTADO = 70% ACIERTOS

Las siguientes pruebas masivas las realizamos con la base de conocimiento obtenida después de aplicar el mantenimiento, es decir, con un conjunto de artículos representativos (aproximadamente 450 artículos).

Antes de realizar las pruebas, nos dimos cuenta de que la aplicación que utilizamos para convertir los artículo de ".PDF" a ".txt" permitía dos opciones de conversión. Dependiendo de la opción elegida se convertía los textos de los artículos de diferente manera. Observamos que este hecho influía en los resultados de la aplicación, ya que dependiendo de la opción seleccionada variaban las 100 primeras líneas del texto de los artículos, que es con lo que trabaja la aplicación. Por ello, decidimos realizar una comparativa entre generar los artículos de la base de conocimiento con una opción de conversión u otra. Los resultados obtenidos fueron:



RESULTADO 1ª Opción de conversión = 85% ACIERTOS

RESULTADO 2ª Opción de conversión = 80% ACIERTOS

Conseguimos aumentar hasta un 80-85% el porcentaje de aciertos, gracias a que con el mantenimiento conseguimos una base de conocimiento más representativa que la base de conocimiento que utilizamos al principio. Decidimos quedarnos con la primera opción de conversión, ya que aunque la diferencia es pequeña, tiene un mayor número de aciertos.

Observamos que en el 90% de los casos en los que la aplicación falla, es decir, que no le asigna al artículo que se está catalogando el mismo *Materia 2* que tiene asignado, por lo menos lo cataloga con el mismo *Materia 1* y con otro *Materia 2* dentro del mismo *Materia 1.*

También hay que tener en cuenta dos cuestiones, la primera es que hemos observado que parece que algunos artículos no están bien catalogados por la Biblioteca. Por ejemplo, un artículo perteneciente a una revista de Biología tiene asignada una *Materia 2* de "Derecho", lo cual no creemos que sea correcto y nuestra aplicación le asigna a ese artículo una *Materia 2* de "Biología", que puede tener más sentido, pero lo anotamos como fallo porque no es la misma materia que tiene asignada por la Biblioteca. La segunda, es el hecho de que algunas veces nuestra aplicación le puede asignar al artículo una *Materia 2* distinta de la que tiene pero que también puede ser correcta, y que nosotros al no ser unos expertos en catalogación también la anotamos como fallo cuando podría ser un acierto. Por lo anteriormente comentado, hemos tenido muchas dificultades a la hora de realizar estas pruebas, debido a los problemas para diferenciar entre aciertos y fallos.

Estos experimentos están referidos sólo a los metadatos *Materia 1 y Materia 2,* el resto de metadatos (Resumen, Palabras Clave, Abstract y Keywords) no se tienen en cuenta en el N-Cross

Validation, ya que como hemos comentado anteriormente estos metadatos se extraen siempre que estén incluidos en el artículo.

Como se puede observar en las siguientes gráficas, hemos conseguido aumentar el tanto por ciento de aciertos de nuestra aplicación en un 15% al utilizar una base de conocimiento más representativa gracias a la aplicación del mantenimiento. La gráfica de la *figura 3.2* se corresponde con la base de conocimiento proporcionada por Laura Henche, mientras que la gráfica de la *figura 3.3* se corresponde con la base de conocimiento obtenido tras aplicar el mantenimiento.

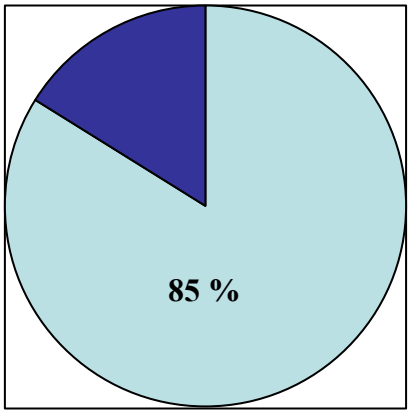


Figura 3.3

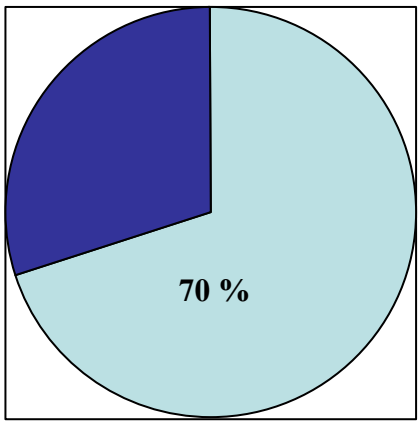


Figura 3.2

4. PROCESO RECOMENDADOR

4.1 Prototipos iniciales

Decidimos realizar varios prototipos de recomendadores utilizando las diferentes opciones que nos ofrecían las plantillas de recomendadores incluidas en el framework jCOLIBRI2. La idea era iniciarnos en el desarrollo de recomendadores, estudiando así sus características, y poder mostrar estos prototipos como un primer acercamiento a este campo, y de esta manera obtener las opiniones de la Biblioteca para poder desarrollar posteriormente un prototipo final más ajustado a sus necesidades. Para simplificar el trabajo, ya que la Biblioteca dispone de múltiples recursos (Portal de Revistas Electrónicas, Catalogo Cisne, E-Prints…), decidimos centrarnos en recomendar artículos del Portal de Revistas Electrónicas de la Biblioteca de la UCM, puesto que estábamos familiarizados con este recurso después de la realización del Catalogador.

A continuación veremos en los siguientes subapartados el desarrollo de los distintos prototipos realizados.

4.1.1 Single-Shot

El primer recomendador que decidimos implementar es uno de los más sencillos, del tipo Single-Shot [8], es decir, de una sola interacción con el usuario. El usuario realiza una consulta (o query) y el sistema le devuelve un conjunto de casos con los 10 artículos más similares a la consulta que ha realizado.

Para explicar el funcionamiento del recomendador seguiremos el esquema que propone jCOLIBRI2 en el método main de la clase principal (*Figura 2.8*). Primero se configura la base de casos, el conector y las funciones de similitud que se van a utilizar (método configure()); luego se ejecuta el preciclo donde se carga la base de casos (método precycle()); a continuación se lanza el ciclo CBR donde se obtiene la consulta, se realiza la recuperación y se muestran los resultados (método cycle()). Por último, se cierra el conector y se liberan recursos (método postcycle()).

También es importante saber que la clase principal (que hereda de la clase StandardCBRApplication) que ejecutará estos métodos en su método principal consta de tres atributos básicos: el conector (clase Connector), la base de casos (clase CBRCaseBase) y un objeto que contendrá las funciones de similitud que se llamará simconfig (clase NNConfig).

El esquema quedaría de la siguiente manera:

```
public class Recomendador implements StandardCBRApplication
{
    /** Conector */
    Connector _connector;

    /** Base de Casos */
    CBRCaseBase _caseBase;

    /** Funciones de similitud */
    NNConfig simConfig;
…

public static void main(String[] args) {

      StandardCBRApplication recommender = new Recomendador();
      try
      {
          recommender.configure();

          recommender.preCycle();

          CBRQuery query = new CBRQuery();

          Articulo_Description hd = new Articulo_Description();
          hd.setApellidos_autor("");
          hd.setNombre_autor("");
          hd.setTitulo("");
          hd.setMateria1(null);
          hd.setMateria2(null);
          hd.setRevista("");
          hd.setKeywords("");

          query.setDescription(hd);

          recommender.cycle(query);
```

```
            recommender.postCycle();

        } catch (Exception e)
        {

org.apache.commons.logging.LogFactory.getLog(Recomendador.class).error(e);

        }
    }
```

Figura 4.1

- **Configure()**

En este método se definen 3 aspectos: cómo va a ser la base de casos, cómo va a ser el conector que relacione la información sobre los artículos con la base de casos y cuáles van a ser las funciones de similitud de cada atributo de los casos. Para este primer recomendador decidimos que la base de casos iba a ser lineal, es decir, que los casos se almacenan en memoria en una estructura de tipo array y el acceso a cada caso se hace a través de un índice. Formalmente, el objeto que representa la base de casos será de la clase LinealCaseBase().

Para crear la base de casos hace falta un conjunto de información que represente a los casos (artículos del Portal de Revistas Electrónicas de la UCM). Para conectar dicha información a la base de casos en memoria se utiliza el conector. En este prototipo utilizamos un conector de la clase PlainTextConnector(). Este tipo de conectores leen línea por línea de un archivo de texto y mapean la información en la base de casos. La manera en que interpreta dicha información se especifica en un archivo xml. El contenido de este archivo (plaintextconfig.xml) para este recomendador es el siguiente:

```
<TextFileConfiguration>

<FilePath>jcolibri/Recomendador_Articulos/database/recomendador.txt</FilePath>
<Delimiters>|</Delimiters>
<DescriptionClassName>jcolibri.Recomendador_Articulos.Articulo_Description</DescriptionClassName>
<DescriptionMappings>
        <Map>titulo</Map>
        <Map>titulo2</Map>
        <Map>Apellidos_autor</Map>
        <Map>Nombre_autor</Map>
        <Map>m1</Map>
        <Map>m2</Map>
        <Map>keywords</Map>
        <Map>keywords2</Map>
        <Map>revista</Map>
</DescriptionMappings>

</TextFileConfiguration>
```

Figura 4.2

Como se puede observar en la *figura 4.2*, en el archivo xml se declaran varios aspectos de configuración. Primero se especifica de qué archivo se va a obtener la información de los artículos, en este caso del archivo "recomendador.txt". Después, se dice qué símbolo se utilizará para separar los distintos atributos de un caso. Así pues, cada línea del archivo de texto representará un caso y será una concatenación de atributos separados por el símbolo " | " . Posteriormente, se especifica qué clase java servirá como representación de la descripción de un caso. En nuestro caso, la clase se llama Articulo_Description() y será donde se mapeen los atributos. Por último, se especifican los atributos que se van a mapear, poniéndolos en el mismo orden en el que se van a leer del archivo de texto "recomendador.txt".

El archivo "recomendador.txt" como ya hemos comentado, incluye en cada línea la información de un caso, separando los distintos atributos mediante el símbolo '|' de la siguiente manera:

titulo|titulo2|apellidos|nombre|materia1|materia2|descriptores|descriptores2|revista|revista2

Llegados a este punto, el sistema ya sabe de donde tiene que obtener la información para crear la base casos y como interpretar dicha información. Lo último que realiza el método configure() es especificar cuáles van a ser las funciones de similitud global y locales. Para el atributo *keywords* (palabras clave) utilizamos la función TokensContained() que compara las palabras de dicho atributo y cuantas más palabras en común haya entre el caso y la consulta mayor será la similitud. Para el resto de atributos utilizamos la función Equal() que devuelve 1 si es igual al valor de la consulta y 0 si es distinto. La similitud global entre el caso y la query se realiza con la función Average(), que hace la media de las funciones locales de cada atributo.

- **Precycle()**

En este método se carga en memoria la base de casos a través del conector que ya sabe cómo debe hacerlo porque se le ha indicado en el método anterior. El conector lee línea a línea el archivo "recomendador.txt", crea el caso correspondiente, mapea los atributos y lo añade a la base de casos. Para esta primera versión utilizamos una base casos de unos 20000 artículos, donde para describir cada artículo hay los siguientes atributos:

- Titulo: título del artículo
- Titulo2: título en inglés del artículo
- Apellidos: apellidos del autor
- Nombre: nombre del autor
- Materia1: materia principal
- Materia2: materia secundaria
- Descriptores: palabras clave que representan al artículo
- Descriptores2: palabras clave en inglés
- Revista: nombre de la revista a la que pertenece el artículo
- Revista2: nombre en inglés de la revista

- **Cycle()**

Este es el método más importante de la aplicación. El esquema que sigue el recomendador es el típico de un sistema Single-Shot como se puede observar en la *figura 4.3*:

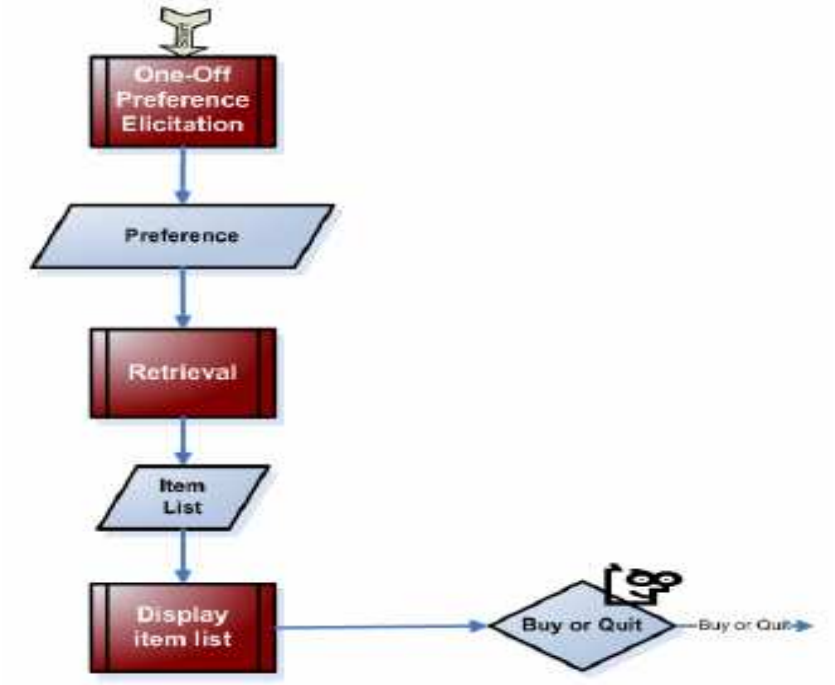


Figura 4.3

Primero se realiza la tarea *One-Off Preference Elicitation* donde se obtiene la consulta del usuario. Esta tarea la implementamos mediante Form-Filling, es decir, rellenando un formulario compuesto por una serie de campos significativos para describir un artículo del Portal de Revistas Electrónicas de la UCM. Los campos que escogimos para el formulario son: título, autores, materia principal, materia secundaria, palabras clave y revista (*Figura 4.4*).

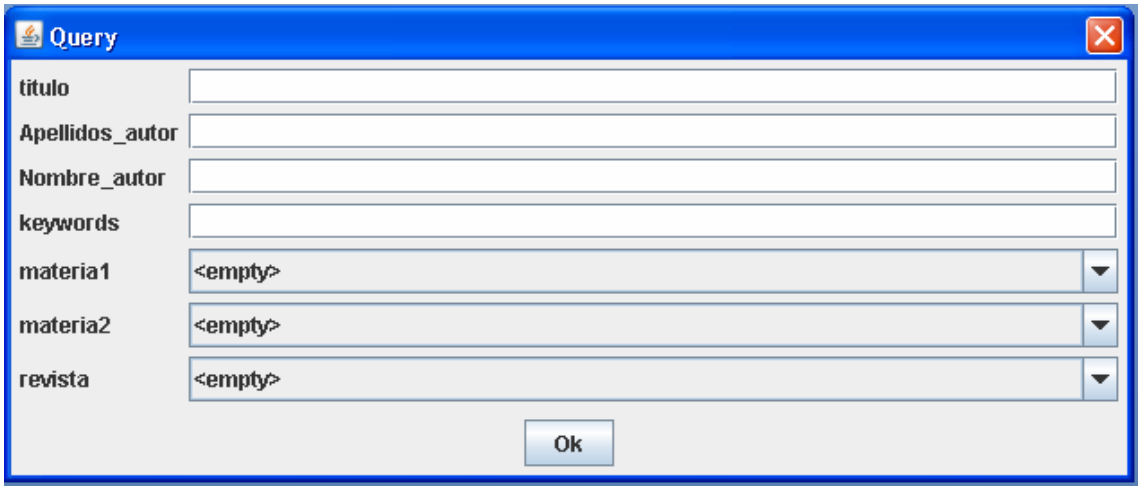


Figura 4.4

No es obligatorio rellenar todos los campos, pero lógicamente cuantos más se rellenen más información tiene el sistema para realizar la recuperación en la base de casos y mejores serán los resultados obtenidos. Estos campos se mapean en un objeto de la clase java CBRQuery (consulta CBR) con lo que ya se tiene la consulta inicial para poder proceder a la recuperación.

Una vez obtenida la consulta, la siguiente tarea es la de recuperación de los casos más similiares a la consulta (*Retrieval*). Para ello utilizamos el algoritmo *KNN* (k-nearest neighbour) que obtiene los k casos más parecidos a la consulta. Este algoritmo se divide en varias fases, primero se evalúa cada caso comparándolo con la query asociándole un valor numérico entre 0 y 1. Esta labor la realiza el método *NNScoringMethod.evaluteSimilarity*. La comparación se realiza con las funciones de similitud que tiene el objeto simconfig que se pasa por parámetro al método anterior y que se configuró anteriormente. Luego se seleccionan los k casos con mayor similitud mediante el método *SelectCases.selectTopK*.

Por último se realiza la tarea *Display item list*, donde se muestran los artículos seleccionados en forma de tabla. En ella se pueden observar los resultados de la recuperación y marcar uno de ellos como solución a la recomendación o salir de la aplicación. Como es un sistema Single-Shot aquí acaba la interacción con el usuario.

- **Postcycle()**

En este método se cierra el conector, se liberan recursos y se finaliza la aplicación.

4.1.2 Single-Shot con Lucene

El segundo prototipo que realizamos fue un recomendador Single-Shot que incorpora el algoritmo de Lucene para realizar la recuperación [4]. Sigue el mismo esquema que el recomendador anterior ya que es del tipo Single-Shot, una sola interacción con el usuario, pero ahora utiliza para la recuperación de casos el algoritmo de Lucene.

Este algoritmo realiza la comparación de textos basándose en el Modelo del Espacio Vectorial [7]. La idea es la siguiente: cada documento, formado por muchas palabras, se representa como un vector en un espacio n-dimensional y la similitud entre documentos se realiza comparando sus correspondientes vectores. El vector tiene tantas componentes como palabras haya en el diccionario y cada componente es el promedio entre la frecuencia de la palabra en el documento y el peso que se le otorga a dicha palabra. Así, documentos que tengan muchas palabras en común tendrán vectores con componentes similares y la similitud entre ellos será mayor. Como este algoritmo es muy dependiente del idioma utilizado existen versiones para diferentes idiomas. Nosotros hemos utilizado la versión en castellano que tiene su propia lista de palabras vacías, método de extracción de raíces, etc. Este algoritmo se ha comprobado que es muy eficiente y da buenos resultados.

Para trabajar con Lucene no nos basta con tener unos cuantos atributos de cada artículo (titulo, autor, palabras clave…) sino que es necesario tener los textos completos de los artículos. Por eso los servicios de la Biblioteca nos facilitaron unos 1200 archivos pdf que contenían los artículos completos.

El esquema del método principal sigue siendo el mismo: configure(), precycle(), cycle() y postcycle(). Los atributos de la clase principal siguen siendo una base de casos, un conector y ahora en vez de un objeto con las funciones de similitud, se tiene un objeto de la clase LuceneIndex que representa un conjunto de documentos normalizados (sin palabras vacías, raíces…) que se utilizarán para la similitud.

- **Configure()**

La tarea de este método es la misma que antes, definir como va a ser la base de casos y el conector. La base de casos es lineal (LinealCaseBase) como en el recomendador anterior, pero el conector es del tipo DataBaseConnector ya que ahora hay que leer de una base de datos en SQL. En el proceso de lectura de una base de datos intervienen varios archivos en los que hay que especificar cómo llevar a cabo el volcado de la base de datos en la base de casos.

En el método *HSQLDBServer.init()* se especifica donde está la base de datos de la que se va a leer. En nuestro caso se trata de un archivo SQL que se llama "clasifica.sql", en el cual aparece una tabla donde cada artículo viene representado por un identificador numérico, el nombre del archivo y la materia2 (en valor numérico) que tiene asociado el artículo. El aspecto de cada fila de la tabla, que representa un documento es el siguiente:

```
INSERT INTO  clasifica VALUES (56, 'ANQE9090110035A.PDF', 711);
```

Este documento tiene como identificador numérico el 56, el nombre del archivo pdf es ANQE9090110035A.PDF y la materia secundaria asociada es la 711 (que corresponde con Filología Árabe).

Una vez se conoce cual es la base de datos de la que se va a leer, el sistema debe saber cómo interpretar dicha información para crear la base de casos. Eso es lo que hace el método *conector.initFromXMlfile* utilizando un archivo xml. El contenido de este archivo es el que aparece en la *figura 4.5*:

```
<DataBaseConfiguration>
        <HibernateConfigFile>jcolibri/Recomendador_Articulos/hibernate.cfg.xml</HibernateCo
         nfigFile>
        <DescriptionMappingFile>jcolibri/Recomendador_Articulos/ArticuloDescription.hbm.xml
        </DescriptionMappingFile>
        <DescriptionClassName>jcolibri.Recomendador_Articulos.Articulo_Description2</Descr
        iptionClassName>
</DataBaseConfiguration>
```

Figura 4.5

Como puede verse este archivo xml referencia a su vez a otros 3 documentos. El primero, "hibernate.cfg.xml", entre otras cosas, especifica cual es el dialecto de la base de datos que se va a utilizar. En nuestro caso el dialecto es SQL. El segundo archivo, "ArticuloDescription.hbm.xml", relaciona la base de datos (tabla clasifica) con la clase Java donde se va a mapear dicha tabla (*Articulo_Description2*). Además asocia el nombre de la columna de la tabla clasifica con el nombre del atributo en el que se quiere mapear. Por ejemplo, en "clasifica.sql" vemos como la segunda columna se corresponde con el nombre del pdf y la columna se llama *fichero,* entonces se debe asociar al atributo correspondiente de la clase *Articulo_Description2* que se llama *nombreFichero*. El contenido de "ArticuloDescription.hbm.xml" es el mostrado en la *figura 4.6*:

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping default-lazy="false">
<classname="jcolibri.Recomendador_Articulos.Articulo_Description2" table="clasifica">

        <id name="id" column="id">
        </id>
        <property name="nombreFichero" column="fichero"/>
        <property name="materia2" column="idMateria2"/>
</class>
</hibernate-mapping>
```

Figura 4.6

El tercer archivo que se referencia en "databaseconfig.xml" es la clase java que representa un artículo, que como ya hemos mencionado, es la clase *articulo_Description2*. Una vez creados el conector y la base de casos, y cargada la configuración, el sistema ya sabe cómo leer la información para rellenar la base de casos.

- **Precycle()**

Este método carga la base de casos en memoria a través del conector, que ya sabe cómo hacerlo pues ha sido configurado anteriormente. Este proceso lo realiza el método *conector.init2()*. Además este método, obtiene las 50 primeras líneas del artículo y las mapea en el atributo contenido de la clase *Articulo_Description2*. Para leer línea por línea el artículo primero debemos convertir el archivo pdf en un txt, para lo que utilizamos la aplicación *pdftotext*.

Ahora tenemos para cada artículo sus 50 primeras líneas, su nombre y la materia2 correspondiente. Por último hay que crear el índice Lucene para poder realizar la similitud entre casos. Esto es lo que hace el método *createLuceneIndex,*

- **Cycle()**

El ciclo de la aplicación empieza con la tarea One-Off Preference Elicitation (ver *figura 4.3*) obteniendo la consulta a través de un formulario sencillo formado por campos que describan el artículo que desea encontrar. El aspecto del formulario con algunos campos rellenos es el mostrado en la *figura 4.7*:

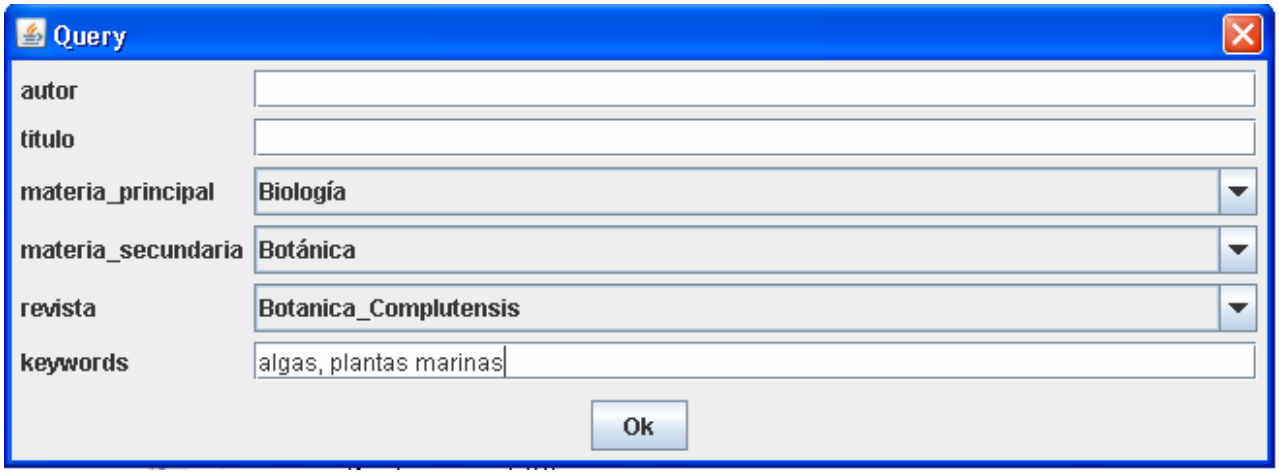


Figura 4.7

Así pues, lo primero es mostrar el formulario al usuario para que lo rellene con su consulta. Esto es llevado a cabo por el método *obtainQueryWithInitialValues* que devuelve un objeto CBRQuery formado por los atributos correspondientes (autor, titulo, materia principal…) con las opciones introducidas por el usuario. En el atributo *contenido* de la CBRQuery se mapea la concatenación de todos los campos del formulario.

Una vez tenemos la query, se realiza la recuperación de los casos más relevantes (*Retrieval*). Para esta tarea utilizamos la técnica de los k-vecinos más próximos (KNN) igual que en recomendador anterior pero esta vez para computar las similitudes entre el caso y la query se utiliza el algoritmo de Lucene sobre el atributo contenido. En los casos este atributo contiene las 50 primeras líneas del texto del artículo correspondiente al caso, mientras que en la query contiene la concatenación de los campos rellenados por el usuario. Después de realizar pruebas con todo el texto, con las 100 primeras líneas y con las 50 primeras líneas, decidimos usar las 50 primeras líneas en vez de utilizar todo el texto del artículo, pues en las primeras líneas suele estar la información suficiente para describirlo (resumen, abstract, palabras resumen, introducción…) y así mejoramos la eficiencia. Además, vimos que en los resultados obtenidos seleccionando las 100 y 50 primeras líneas no había diferencias significativas, por lo que optamos por el menor número de líneas para un mejor rendimiento.

De esta forma, se comparan los casos con la query usando Lucene, y se seleccionan los 10 casos más similares que se muestran al usuario en forma de tabla donde aparece el nombre archivo, la materia2 asociada y el texto de las 50 primeras líneas (*Figura 4.8*).

Figura 4.8

El usuario puede seleccionar uno de los artículos para verlo o salir de la aplicación. Si el usuario decide ver un artículo puede visualizarlo pulsando el botón *Ver Artículo.* Si después desea realizar otra consulta debe hacer clic en *Nueva Recomendación*.

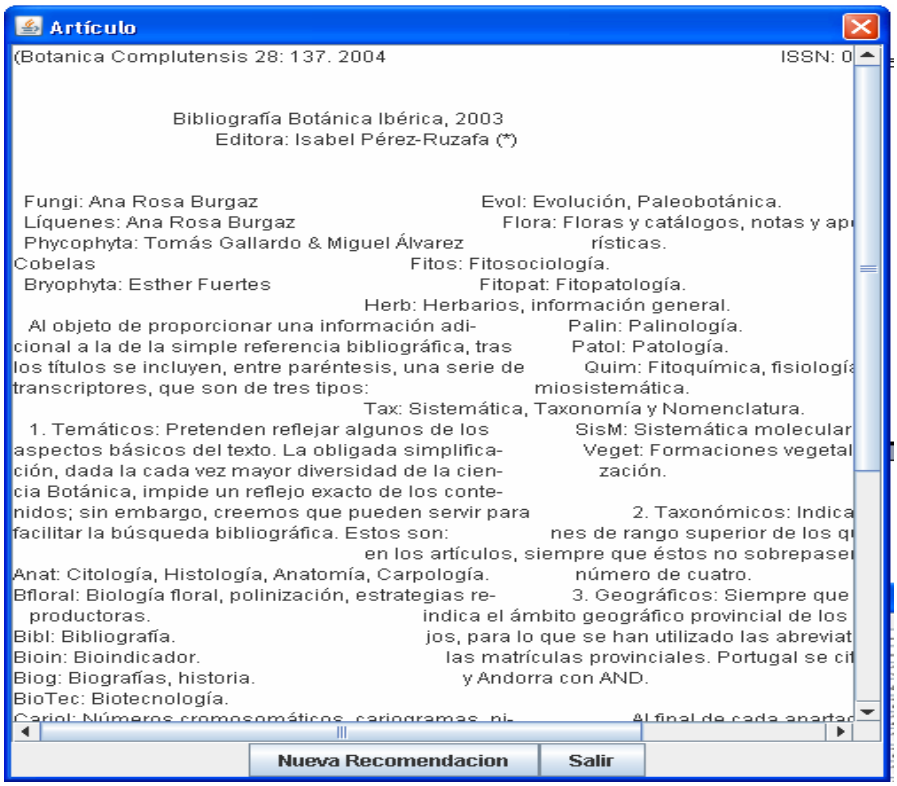


Figura 4.9

- **Postcycle()**

  Este método cierra el conector de manera segura y cierra la aplicación.

4.1.2.1 Experimentos prototipos Single-Shot

Una vez explicado el funcionamiento de los dos prototipos de Recomendadores Single-Shot vamos a mostrar las resultados de las pruebas realizadas a los mismos. Así se podrá hacer una comparativa de ambos sistemas y decidir cuál ofrece mejores resultados. Las medidas que utilizamos para realizar las pruebas son tres: precisión, recall y el tiempo de ejecución de la aplicación.

La precisión de un sistema recomendador se define como:

$$\text{Precisión} = \frac{\text{nº documentos relevantes recuperados}}{\text{nº total de documentos recuperados}}$$

Esta medida es bastante intuitiva. Cuantos más documentos relevantes aparezcan en la recuperación de casos, mayor es la precisión. El caso mejor se da cuando todos los documentos recuperados son relevantes, entonces la precisión es del 100%. Esta medida es interesante pero debe complementarse con el recall, pues si se recuperan muy pocos casos (1 ó 2) la precisión puede ser muy alta pero el recall será muy bajo.

El recall de un sistema recomendador se define como:

$$\text{Recall} = \frac{\text{nº documentos relevantes recuperados}}{\text{nº documentos relevantes en la colección}}$$

El recall (o exhaustividad en español) mide la cantidad de documentos relevantes que se recuperan en función de todos los documentos relevantes en la base de casos. Si se recuperan muy pocos casos relevantes, el recall será muy bajo, pero en cambio si se recuperan muchos, el recall será alto. Esta medida también puede ser engañosa, pues si en el caso extremo se recuperan todos los casos de la colección (sean relevantes o no) el recall será del 100% forzosamente pero se perderá precisión. Así pues, para tener una visión real del rendimiento del sistema se debe hacer una comparativa entre precisión y recall.

Otra aspecto importante para estas medidas es cómo se define el concepto de "documento relevante". Este aspecto es totalmente dependiente del dominio de aplicación. En nuestro caso hemos decidido que un documento es relevante si tiene la misma materia secundaria que la consulta. Así pues, para realizar las pruebas definimos una query fija (con una materia secundaria concreta), realizamos un método para contar el número de casos con esa materia secundaria (y saber cuántos son relevantes) y obtuvimos estos resultados.

Prototipo1 Single-Shot con comparación por campos:

	PRECISIÓN	RECALL
Query1	100%	13%
Query2	90%	100%
Query3	40%	80%
Query4	100%	83%
Query5	100%	100%
Query6	100%	90%
Query7	100%	83%
Query8	45%	60%
Query9	58%	23%
Query10	80%	55%
MEDIA	**81%**	**68%**

Prototipo2 Single-Shot con Lucene:

	PRECISIÓN	RECALL
Query1	90%	36%
Query2	30%	43%
Query3	30%	100%
Query4	90%	28%
Query5	100%	25%
Query6	90%	50%
Query7	100%	20%
Query8	80%	75%
Query 9	66%	93%
Query 10	84%	80%
MEDIA	**75%**	**55%**

Se puede observar como el prototipo1 que realiza comparación por campos tiene mejores resultados que el prototipo2 que utiliza Lucene. Esto es debido a que Lucene se basa en el Modelo del Espacio Vectorial y esta pensado para comparar textos grandes, con muchas palabras. En cambio la consulta que se obtiene, que es la concatenación de todos los campos del formulario en un solo string, es un texto muy pequeño comparado con las 50 primeras líneas de cada artículo, por ello los resultados son un poco peores. Aún así, los resultados de ambos recomendadores son aceptablemente buenos.

La tercera medida que hemos utilizado es el tiempo de ejecución de la aplicación. Hemos dividido dicho tiempo en dos: el tiempo que tarda en cargar la base de casos y el tiempo que tarda en hacer la recuperación. En un sistema recomendador que interactúa con el usuario es importante que el tiempo de recuperación sea pequeño, mientras que el tiempo en cargar la base de casos no es tan importante, puesto que la base de casos podría estar cargada en memoria antes de que el usuario utilice la aplicación. Los resultados obtenidos para ambos prototipos son los siguientes.

Prototipo1 Single-Shot con comparación por campos:

Tiempo en cargar base de casos = 10 segundos

Tiempo en recuperación de casos = 14 segundos

Prototipo2 Single-Shot con Lucene:

Tiempo en cargar base de casos = 5 minutos

Tiempo en recuperación de casos = 25 segundos

Observando los resultados de tiempos se puede concluir que ambos prototipos son demasiado lentos, especialmente el prototipo2, y que necesitan modificaciones que mejoren su rendimiento, sobre todo el tiempo de recuperación de casos.

4.1.3 Conversacional

Para el tercer prototipo de sistema recomendador decidimos cambiar el esquema Single-Shot de los dos recomendadores anteriores y desarrollar un recomendador de tipo Conversacional. La principal diferencia entre ambos esquemas es que en los recomendadores conversacionales existe una mayor interacción entre el usuario y el sistema. En estos sistemas el usuario hace una consulta inicial, el sistema le recomienda una selección de casos y el usuario critica uno o varios de los casos que le propone el sistema, y esa información combinada con la consulta inicial sirve de nueva consulta sobre la que se realiza el ciclo CBR de nuevo. De esta manera, el usuario va refinando la consulta inicial, observando los resultados del recomendador y escogiendo entre los casos que más se parecen a lo que está buscando. Puede repetirse el ciclo CBR tantas veces como el usuario desee. El esquema que hemos utilizado para este recomendador es el mostrado en la *figura 4.10*.

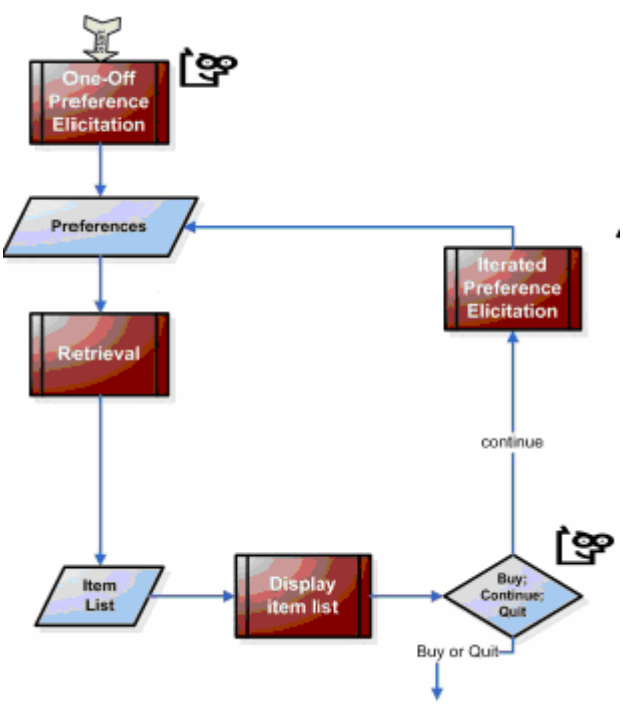


Figura 4.10

Las tareas *One-Off Preference Elicitation* (obtiene consulta inicial), *Retrieval* (recuperación de casos) y *Display Item List* (muestra tabla resultados) son básicamente iguales que las de un sistema Single-Shot, como en los dos recomendadores anteriores. La nueva tarea que proporciona una interacción entre el usuario y el sistema es la tarea *Iterated Preference Elicitation*. En ella, el usuario recibe unos resultados de la recuperación anterior y realiza una crítica, produciendo una nueva consulta que vuelve a repetir el bucle. Al igual que las tareas anteriores, esta tarea admite varias posibilidades para su desarrollo.

Una opción de desarrollo es realizar la nueva consulta mediante Form-Filling, es decir, rellenando un formulario. Cuando has utilizado un formulario también para la consulta inicial lo normal es que ahora puedas modificar los campos en función de los casos que el sistema te ha recuperado anteriormente. Otra opción es la de Navigation-by-Asking donde el usuario responde a unas preguntas que el sistema le plantea según las consultas anteriores y sus preferencias. Es habitual el uso de heurísticas para determinar cual es la mejor pregunta a realizar al usuario. La tercera forma de realizar la tarea *Iterated Preference Elicitation* es mediante el método Navigation-by-Proposing, que es la opción que hemos utilizado para este prototipo. Se basa en que al usuario se le muestran un conjunto de casos candidatos, el usuario escoge el que más se acerca a sus necesidades y plantea una crítica (del tipo “parecido a esto pero…”). El sistema combina el caso seleccionado con la crítica y con las consultas anteriores en una nueva consulta que se utilizará para volver a hacer recuperación de casos.

En cualquiera de las tres opciones el usuario va perfeccionando su consulta con la ayuda del sistema que le proporciona información sobre los casos. Además, el propio proceso de recomendación también mejora según el usuario proporciona más información a la consulta. Es posible pensar que este esquema iterativo produce mejores resultados que los sistemas de una sola vuelta pero a cambio se necesita más tiempo (más interacciones sistema-usuario).

Una vez introducido el esquema del prototipo conversacional explicaremos brevemente la implementación del mismo. Los atributos globales del recomendador son los mismo de antes (una

base de casos, un conector y un objeto de configuración de similitudes) más algunos nuevos (un filtro y una estrategia). El filtro descarta casos que no se parecen estructuralmente a la consulta para no computar la similitud completa entre toda la base de casos y la query. La estrategia define el tipo de crítica que va a poder realizar el usuario (*Figura 4.11*).

```
public class Recomendador3 implements StandardCBRApplication
{
    enum QueryElicitationStrategy {MLT, pMLT, wMLT, LLT, MLT_LLT};

    /** Conector */
    Connector _connector;

    /** Base de Casos */
    CBRCaseBase _caseBase;

    /** Configuración KNN */
    NNConfig simConfig;

    /** Filter  de configuracion */
    FilterConfig filterConfig;

    /** Estrategia */
    QueryElicitationStrategy strategy;

    /** Booleano para nueva recomendación*/
    boolean respuesta;
```

Figura 4.11

- **Configure()**

El primer método que se lanza es el de crear y configurar los objetos principales. Se crea una base de casos lineal y un conector a texto plano, pues la fuente de información vuelve a ser un documento de texto ("recomendador.txt") con una línea por artículo y distintos atributos (titulo, autor, materia principal…) separados por un delimitador '|'. El conector se inicializa desde un xml ("plaintextconfig.xml") que le indica dónde está la fuente de información, cuales son los delimitadores, cómo se corresponden los campos de una línea con los atributos de cada artículo y qué clase java representa la descripción de un artículo (clase *Articulo_Description*) para mapear dichos campos en los atributos correspondientes. También se crea la configuración de similitudes (objeto simConfig) y para cada atributo de la descripción de un artículo se escoge una función de similitud. Se crea el filtro y se escoge la estrategia con la que vamos a trabajar. En nuestro caso hemos optado por la estrategia "More-Like-This" (parecido a éste).

- **Precycle()**

En el preciclo se carga la base casos a través del conector. Éste ya sabe de dónde extraer la información y cómo interpretarla, pues ha sido configurado anteriormente. Se lee del documento "recomendador.txt" los metadatos (título, titulo en inglés, apellidos autor, nombre autor, materia principal, materia secundaria, descriptores, descriptores en inglés, revista y revista en inglés) de unos 20000 artículos. Se mapea cada línea en la descripción de un objeto CBRCase y se obtiene la base de casos. Hay que tener en cuenta que no siempre están rellenos todos los campos.

- **Cycle()**

Una vez tenemos la base de casos cargada en memoria se puede proceder a ejecutar el ciclo CBR que es la parte fundamental del sistema. Para ello se sigue el esquema de un recomendador Conversacional dividido en tareas.

Lo primero es obtener la consulta inicial del usuario, llevada a cabo por la tarea *One-Off Preference Elicitation*. Se realiza mostrando un formulario que el usuario rellena (Form-Filling). Los campos del formulario se corresponden con los atributos que forman la descripción del artículo, por ello la recuperación separada por campos es la más interesante. En la *figura 4.12* mostramos una consulta de ejemplo.

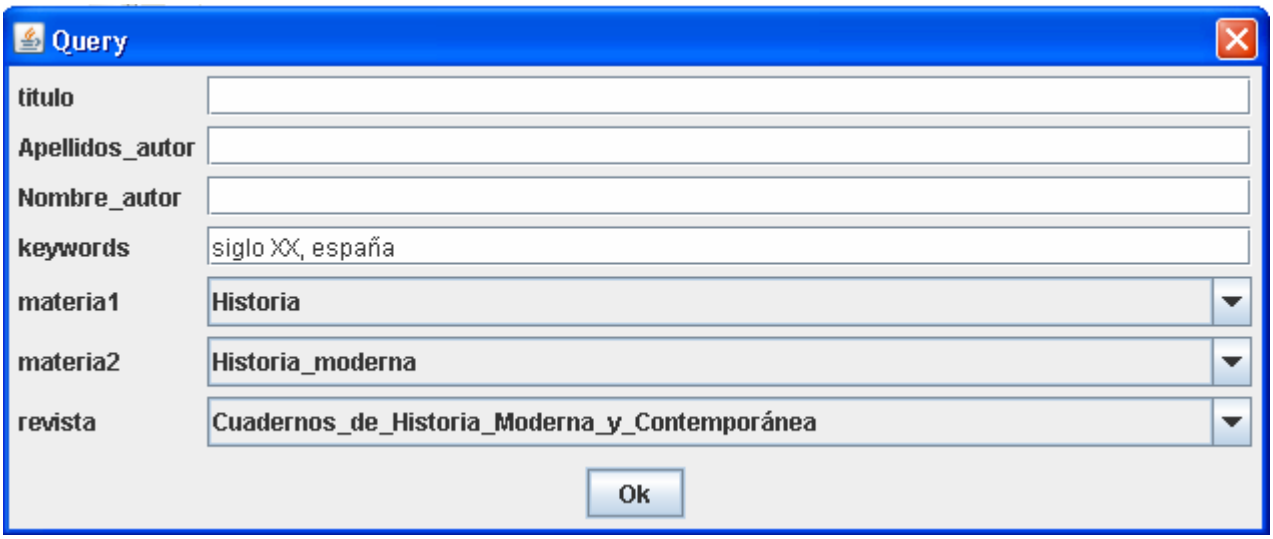


Figura 4.12

La siguiente tarea del ciclo es la de recuperación de los casos relevantes (*Retrieval*). Utiliza el método KNN con las funciones de similitud asignadas en la configuración. Compara cada atributo de la consulta con el mismo atributo de cada caso de la base de casos mediante la función de similitud correspondiente, y luego hace la media de todos los atributos para obtener una medida de similitud global entre el caso y la consulta. Por último, se seleccionan los 10 casos más similares y se muestran por pantalla en forma de tabla (*Figura 4.13*).

Figura 4.13

Ahora llega la nueva tarea del sistema (*Iterated Preference Elicitation*) donde el usuario refina la consulta anterior. Nosotros lo hacemos mediante Navigation-by-Proposing como ya

hemos comentado anteriormente. El sistema le propone 10 casos similares a la consulta y el usuario escoge el que más se acerque a sus gustos y lo critica seleccionándolo y pulsando el botón "Something Like This". El sistema recibe esta información y reformula la consulta. La manera en que combina la query anterior con la actual es concatenando la nueva información de cada campo al valor anterior. De esta forma, no se pierde información sobre la primera consulta y cada vez se tiene una consulta más completa para la nueva recuperación. Esta forma de recomendar es aconsejable cuando el usuario no sabe bien lo que está buscando o inicialmente tiene poca información. En una primera consulta puede sólo rellenar un campo (por ejemplo Biología como materia principal) y según los casos que le recupere el sistema, el usuario puede ir completando su consulta (por ejemplo marcando un artículo con materia secundaria Botánica).

- **Postcycle()**

  Este método cierra el conector correctamente y termina la aplicación.

4.1.3.1 Experimentos prototipo Conversacional

Para un sistema recomendador conversacional las pruebas son distintas a las de un sistema Single-Shot. Al ser un sistema iterativo, en el que hay varias interacciones entre el usuario y el sistema, en lugar de medir la precisión y el recall en una vuelta, las medidas se centran en el número de iteraciones que necesita el sistema hasta encontrar una query fijada de antemano.

Así pues, se debe escoger un artículo de la base de casos para utilizarlo como query para el recomendador. Se introduce dicho artículo como consulta y se cuenta el número de pasos hasta que el sistema propone dicho artículo como solución. A continuación mostramos los resultados:

	NÚMERO DE PASOS
Query1	4
Query2	2
Query3	2
Query4	3
Query5	5
Query6	2
Query7	3
Query8	No encontrada (>10 pasos)
Query9	1
Query10	3
MEDIA	**2.7**

Se puede observar como en la mayoría de los casos tarda 2 ó 3 vueltas en encontrar la query como solución, lo cuál indica un buen funcionamiento del sistema. Hay que decir que las querys utilizadas tienen más de la mitad de los campos rellenos en el formulario, por lo que se trata de consultas bastante concretas. También hay una vez que el sistema no encuentra la consulta en la base de casos aunque se trate de un caso que hemos extraído de la misma, lo que indica que el sistema se pierde en la recuperación o que los casos seleccionados con la estrategia MoreLikeThis no son los adecuados para llegar a la consulta inicial.

Otra medida de evaluación es contar el número de pasos en encontrar una solución establecida "a priori" tomando como query un mismo caso de la base de casos, pero realizando esta prueba varias veces variando la información de la query de menos específica a más específica. Es una variación de la medida anterior, solo que ahora se toma un mismo caso como consulta y se repite el proceso de recomendación con distintas querys del mismo caso. Es decir, se escoge un artículo de la base de casos como consulta, se hace una primera recomendación con una query que sólo tenga un campo relleno, luego se hace otra recomendación con dos campos rellenos del mismo artículo, luego con tres campos rellenos y así sucesivamente hasta que en la última recomendación se copian todos los campos del artículo en la query. Los resultados obtenidos para un artículo dado son:

Campos rellenos en la query	Número de pasos
1	6
2	6
3	4
4	3
5	2
6	1
7	1

Como cabía esperar cuanto más específica es la consulta menos pasos necesita el recomendador para encontrar el artículo en cuestión. También es interesante la importancia de rellenar unos campos u otros. Hemos observado que la materia secundaria es un atributo muy relevante en la recuperación y cuando rellenas dicho campo se encuentra el artículo en menos pasos. Las palabras clave en cambio no resultan tan decisivas.

Por último, la otra medida de evaluación del rendimiento es el tiempo de ejecución de la aplicación. En este prototipo distinguimos dos tiempos: el tiempo en cargar la base de casos y el tiempo en recuperar los casos más similares en cada vuelta. Los resultados en media son:

Tiempo en cargar base de casos = 5 minutos

Tiempo en recuperación de casos = 25 segundos

4.1.4 Colaborativo

Pensamos que sería útil desarrollar un recomendador que tuviera en cuenta los perfiles de cada usuario y sus preferencias (artículos favoritos), y por ello, decidimos realizar este prototipo.

La idea es que para que un usuario pueda utilizar este servicio debe haberse registrado previamente rellenando un perfil de usuario. Este perfil de usuario será utilizado posteriormente por la aplicación a la hora de recomendarle artículos del Portal de Revistas Electrónicas. También vimos conveniente, que la aplicación no sólo recomendara en función de los perfiles de usuario sino que también tuviera en cuenta los artículos favoritos de cada usuario, de forma que cada usuario pueda agregar artículos a una lista de artículos favoritos con una puntuación para cada uno.

En general, el recomendador que diseñamos tenía en cuenta los perfiles de usuario junto con su lista de artículos favoritos y sus puntuaciones para realizar las recomendaciones. La interacción del sistema recomendador sería la siguiente:

- La primera vez que el usuario accede a la aplicación debe rellenar su perfil de usuario.
- Cada vez que el usuario vuelve a utilizar la aplicación debe registrarse, y después de hacerlo el sistema le recomienda un conjunto de artículos. El usuario puede consultar los artículos recomendados u otros artículos.
- El usuario puede añadir cualquier artículo a su lista de favoritos añadiéndole una puntuación. Cuantos más artículos favoritos tenga el usuario mejor le recomendará la aplicación.

Siguiendo estas ideas vimos que el primer paso a realizar era el de especificar los campos que se iban a incluir en el perfil de usuario, para ello nos reunimos con Laura Henche (Programadora del Dpto. de Automatización de la Biblioteca) y con ella llegamos a un acuerdo sobre cuales serían los campos más convenientes para el perfil de usuario. Decidimos que el perfil iba a estar orientado a alumnos y profesores, ya que pensamos que serían los principales usuarios de esta aplicación. De esta forma, el perfil de usuario quedó inicialmente especificado de la siguiente manera:

- Email: Identifica a cada usuario.

- Facultad: Representa el centro en el que estudia o trabaja el usuario.

- Materia 1: Es la materia principal favorita del usuario. Existen 32 materias entre las que se encuentran "Biología", "Física", "Historia"…

- Materia 2: Es la materia secundaria o más específica favorita del usuario. Por ejemplo la *Materia 1* "Biología" contiene como *Materias 2* específicas: "Botánica", "Biología Marina"…

- Idioma: Representa el idioma que prefiere el usuario.

Una vez especificados los campos del perfil de usuario, nos centramos en el estudio de las similitudes entre perfiles de usuario. Trabajamos con dos tipos de funciones de similitud, la primera es la función de similitud global que se encarga de obtener la similitud final entre perfiles de usuario, y la segunda es la función de similitud local propia de cada campo del perfil. Ambos tipos de funciones devuelven un valor de similitud comprendido entre "0" y "1". Como función de similitud global utilizamos la función *Average,* que asigna a cada campo del perfil el mismo peso o importancia para el cálculo final de la similitud. Como funciones de similitud locales, inicialmente utilizamos la función *Equal,* que devuelve "1" si el campo que se compara de un perfil es igual al mismo campo del otro perfil con el que se está realizando la similitud. Observamos que esta función de similitud local era apropiada para el campo Idioma ya que entre distintos idiomas no existe ninguna relación, pero en cambio, era demasiado simple para los campos Facultad y Materia 1. Por ejemplo, si un usuario tiene como Facultad "Biología", es probable que tenga más artículos en común o gustos más parecidos con otro usuario de "Geología" que con otro que sea de "Derecho". Por ello, desarrollamos unas tablas de similitud (Tabla Facultades y Tabla Materias 1) para utilizar la función de similitud local *Table,* que se encarga de obtener la similitud de estos campos a partir de una tabla.

En la *figura 4.14* se puede observar la Tabla Facultades. El criterio que utilizamos para generar esta tabla y de forma análoga la Tabla Materias 1 fue el siguiente:

- Se le asigna similitud "1" si es la misma Facultad.
- Se le asigna similitud "0.7" si las dos facultades pertenecen a la misma área (Ciencias o Letras) y además están muy relacionadas como por ejemplo "Medicina" y "Enfermería" o "Matemáticas" y "Estadística".
- Se le asigna similitud "0.5" si las dos facultades pertenecen a la misma área (Ciencias o Letras), como por ejemplo "Informática" y "Biología".
- Se le asigna similitud "0.1" si las dos facultades no pertenecen a la misma área, como por ejemplo "Física" e "Historia del Arte".

En esta tabla, los nombres de los centros o facultades están abreviados de la siguiente manera:

- Bba → Facultad de Bellas Artes
- Bio → Facultad de Ciencias Biológicas
- Byd → Facultad de Ciencias de la Documentación
- Cca → Instituto Universitario de Ciencias Ambientales
- Cee → Facultad de Ciencias Económicas y Empresariales
- Cps → Facultad de Ciencias Políticas y Sociología
- Der → Facultad de Derecho
- Edu → Facultad de Educación
- Eis → Facultad de Informática
- Emp → Escuela Universitaria de Estudios Empresariales
- Enf → Escuela Universitaria de Enfermería y Fisioterapia
- Est → Escuela Universitaria de Estadística
- Far → Facultad de Farmacia
- Fis → Facultad de Ciencias Físicas
- Fll → Facultad de Filología
- Fsl → Facultad de Filosofía
- Geo → Facultad de Ciencias Geológicas
- Ghi → Facultad de Geografía e Historia
- Inf → Facultad de Ciencias de la Información
- Mat → Facultad de Ciencias Matemáticas
- Med → Facultad de Medicina
- Odo → Facultad de Odontología
- Opt → Escuela Universitaria de Óptica
- Psi → Facultad de Psicología
- Qui → Facultad de Ciencias Químicas
- Trs → Escuela Universitaria de Trabajo Social
- Vet → Facultad de Veterinaria

	Bba	Bio	Byd	Cca	Cee	Cps	Der	Edu	Eis	Emp	Enf	Est	Far	Fis	Fll	Fsl	Geo	Ghi	Inf	Mat	Med	Odo	Opt	Psi	Qui	Trs	Vet
Bba	1	0.1	0.5	0.1	0.1	0.5	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.1	0.5	0.5	0.1	0.1	0.1	0.1	0.5	0.1	0.5	0.1
Bio	0.1	1	0.1	0.7	0.5	0.1	0.1	0.5	0.5	0.10	0.5	0.5	0.5	0.5	0.1	0.1	0.7	0.1	0.1	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.5
Byd	0.5	0.1	1	0.1	0.1	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.1	0.5	0.7	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.1
Cca	0.1	0.7	0.1	1	0.5	0.1	0.1	0.5	0.1	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.7	0.1	0.1	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.5
Cee	0.1	0.5	0.1	0.5	1	0.1	0.1	0.5	0.5	0.7	0.5	0.7	0.5	0.5	0.1	0.1	0.5	0.1	0.1	0.7	0.5	0.5	0.5	0.1	0.5	0.1	0.5
Cps	0.5	0.1	0.5	0.1	0.1	1	0.7	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.1	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.1
Der	0.5	0.1	0.5	0.1	0.5	0.7	1	0.5	0.1	0.5	0.1	0.1	0.1	0.1	0.5	0.5	0.1	0.5	0.7	0.1	0.1	0.1	0.1	0.5	0.1	0.5	0.1
Edu	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Eis	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	1	0.5	0.5	0.5	0.1	0.7	0.1	0.1	0.5	0.1	0.1	0.7	0.5	0.5	0.5	0.1	0.5	0.1	0.5
Emp	0.1	0.5	0.1	0.1	0.7	0.7	0.5	0.5	0.7	1	0.5	0.7	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.7	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Enf	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.1	0.1	1	0.1	0.7	0.5	0.1	0.1	0.5	0.1	0.1	0.5	0.7	0.7	0.7	0.7	0.5	0.1	0.7
Est	0.1	0.5	0.1	0.5	0.7	0.1	0.1	0.5	0.5	0.7	0.5	1	0.5	0.5	0.1	0.1	0.5	0.1	0.1	0.7	0.5	0.5	0.5	0.1	0.5	0.1	0.1
Far	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.5	0.1	0.7	0.5	1	0.5	0.1	0.1	0.5	0.1	0.1	0.5	0.7	0.7	0.7	0.5	0.7	0.1	0.5
Fis	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.7	0.5	0.5	0.5	0.5	1	0.1	0.1	0.5	0.1	0.1	0.7	0.5	0.5	0.5	0.1	0.7	0.1	0.5
Fll	0.5	0.1	0.5	0.1	0.1	0.5	0.1	0.5	0.1	0.1	0.1	0.1	0.1	0.1	1	0.5	0.1	0.5	0.5	0.1	0.1	0.1	0.1	0.5	0.1	0.5	0.1
Fsl	0.5	0.1	0.5	0.1	0.1	0.5	0.1	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	1	0.1	0.5	0.5	0.1	0.1	0.1	0.1	0.5	0.1	0.5	0.1
Geo	0.1	0.7	0.1	0.7	0.5	0.1	0.1	0.5	0.5	0.1	0.5	0.5	0.5	0.5	0.1	0.1	1	0.1	0.1	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.5
Ghi	0.5	0.1	0.5	0.1	0.1	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.1	1	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.1
Inf	0.5	0.1	0.5	0.1	0.1	0.5	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.1	0.5	1	0.1	0.1	0.1	0.1	0.5	0.1	0.5	0.1
Mat	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.7	0.5	0.5	0.5	0.5	0.7	0.1	0.1	0.5	0.1	0.1	1	0.5	0.5	0.5	0.1	0.7	0.1	0.5
Med	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.1	0.1	0.7	0.1	0.7	0.5	0.1	0.1	0.5	0.1	0.1	0.5	1	0.7	0.7	0.7	0.5	0.1	0.7
Odo	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.1	0.1	0.7	0.1	0.7	0.5	0.1	0.1	0.5	0.1	0.1	0.5	0.7	1	0.7	0.7	0.5	0.1	0.7
Opt	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.1	0.1	0.7	0.1	0.7	0.5	0.1	0.1	0.5	0.1	0.1	0.5	0.7	0.7	1	0.7	0.5	0.1	0.7
Psi	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.1	0.5	0.1	0.5	0.1	0.1	0.5	0.1	0.1	0.1	0.1	0.5	0.5	0.5	1	0.1	0.5	0.1
Qui	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.7	0.5	0.5	0.5	0.5	0.7	0.1	0.1	0.5	0.1	0.1	0.7	0.5	0.5	0.5	0.1	1	0.1	0.5
Trs	0.5	0.1	0.5	0.1	0.1	0.5	0.5	0.5	0.1	0.5	0.1	0.1	0.1	0.1	0.5	0.5	0.1	0.5	0.5	0.1	0.1	0.1	0.1	0.5	0.1	1	0.1
Vet	0.1	0.5	0.1	0.5	0.5	0.1	0.1	0.5	0.1	0.1	0.7	0.1	0.7	0.5	0.1	0.1	0.5	0.1	0.1	0.5	0.7	0.7	0.7	0.7	0.5	0.1	1

Figura 4.14 – Tabla Facultades

En el caso del campo Materia 2, no era viable la elaboración de una tabla como las anteriores, ya que existen aproximadamente 750 materias 2 diferentes. Por esta razón, decidimos seguir utilizando la función de similitud local *Equal.*

Como se ha comentado anteriormente, a la hora de recomendar a parte de hacer la similitud con los perfiles de usuario, se tienen también en cuenta los artículos favoritos de cada usuario y la puntuación asignada a cada uno de ellos. Para realizar un recomendador que implemente todas estas ideas, nos basamos en el Recomendador Colaborativo (Movies Recommender) implementado en el framework jCOLIBRI2.

Este tipo de recomendador necesita una base de casos específica, en la cual los casos estén compuestos por una descripción, una solución y un resultado. La descripción contiene la información acerca del usuario (perfil de usuario), la solución almacena la información acerca del artículo favorito del usuario, y por último el resultado guarda la puntuación del artículo favorito almacenado en la solución. En la *figura 4.15* se puede observar como sería la estructura de un caso.

Descripción	**Solución**	**Resultado**
Email	Título	Puntuación
Facultad	IdOrigen	
Materia1	IdAjena	
Materia2		
Idioma		

Figura 4.15

Como se puede apreciar en la *figura 4.15,* la descripción está formada por los campos del perfil de usuario que ya hemos comentado anteriormente. La solución está compuesta por el título del artículo y por dos campos más: IdOrigen e IdAjena. Estos dos campos nos sirven para poder generalizar nuestro recomendador de forma que sea capaz de recomendar documentos pertenecientes a otros recursos de la Biblioteca, como por ejemplo: Tesis, e-Prints, Catálogo Cisne… Nuestra aplicación, al ser un prototipo inicial se centra como hemos comentado anteriormente en recomendar artículos del Portal de Revistas Electrónicas por lo que el IdOrigen es siempre "3" y el IdAjena se corresponde con el nombre asignado al artículo. El resultado contiene únicamente el campo puntuación que es un entero comprendido entre "1" y "5", siendo "5" la máxima puntuación asignable.

Por lo tanto, la implementación de la base de casos almacena los casos en una tabla con la siguiente estructura (*figura 4.16*):

	Item1	Item2	Item3	Item4	Item5	...	ItemM
User1		rating12		rating14			
User2	rating21		rating23				rating2N
User3			rating33	rating34			rating3N
...							
UserN		ratingN2			ratinN5		ratingNN

Figura 4.16

Los "User" se corresponden con los usuarios, los "Item" con los artículos y los "ratings" con las puntuaciones de los usuarios a los artículos.

Esta base de casos permite la existencia de diferentes casos que contengan la misma descripción, ya que un usuario puede tener varios artículos favoritos puntuados.

El comportamiento del recomendador se divide en tres fases:

1. Computar la similitud de todos los usuarios respecto al usuario sobre el que se va a realizar la recomendación.
2. Seleccionar un subconjunto de usuarios más parecidos.
3. Normalizar las puntuaciones y seleccionar un conjunto de artículos a recomendar basándose en las puntuaciones de los usuarios seleccionados en la fase 2.

En la fase 1, para calcular la similitud entre los usuarios utilizamos la fórmula *Pearson Correlation,* en la que hemos añadido un nuevo factor (*sim´(a,u)*) que calcula la similitud entre dos perfiles de usuario. Esta fórmula se utiliza para obtener la similitud entre cada para de usuarios, quedando de la siguiente manera:

$$sim(a,u) = \frac{\sum_{i=1}^{m}(r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sigma_a \sigma_b} \times \frac{s}{f} \times sim'(a,u)$$

Donde:

- *a, u:* son los usuarios a comparar
- *m:* número de artículos en común
- $\bar{r}$: valor mínimo
- σ : derivación estándar
- *S/f:* factor que hace decrecer la similitud entre usuarios que tienen menos de *f* artículos en común

La fase 2 y la fase 3 son realizadas por el método *CollaborativeRetrievalMethod* implementado en el framework jCOLIBRI2.

Para cargar los casos en la base de conocimiento nos encontramos con varios problemas. Uno de ellos es que como es una aplicación nueva en la Biblioteca no disponían de tablas con una

estructura bien definida y con datos reales que nos sirvieran para crear la base de casos, por lo que tuvimos que generar un fichero de texto de donde se cargaría la base de conocimiento. Además, como ya hemos comentado anteriormente, debido al hecho de que es una aplicación nueva no teníamos datos para realizar pruebas reales, por ello nos tuvimos que crear un conjunto inicial de datos para poder probar la aplicación.

A continuación, vamos a comentar algunos de los experimentos que realizamos para comprobar el buen funcionamiento del recomendador.

4.1.4.1 Experimentos

Para realizar los experimentos nos creamos un conjunto de prueba de diez usuarios, cada uno de ellos con su perfil y sus artículos favoritos. En la *figura 4.17* mostramos tres de estos diez usuarios que tienen perfiles muy parecidos y con artículos en común.

Usuario	Facultad	Materia1	Materia2	Idioma	Artículo	Rating
José	Bio	3	55	3	Aeropalino...	4
José	Bio	3	55	3	Análisis...	3
José	Bio	3	55	3	Las faunas...	5
Jorge	Bio	3	55	3	Primer cata...	4
Jorge	Bio	3	55	3	Aportación...	3
Jorge	Bio	3	55	3	The volatile...	4
Jorge	Bio	3	55	3	Análisis...	3
Jorge	Bio	3	55	3	Las faunas...	5
Cristina	Geo	13	315	3	Las faunas...	5
Cristina	Geo	13	315	3	Aeropalino...	4
Cristina	Geo	13	315	3	Floristic...	4
Cristina	Geo	13	315	3	Evaluación...	3
Cristina	Geo	13	315	3	Tratamiento...	4

Figura 4.17

Tras realizar varias pruebas, nos dimos cuenta de que era necesario para obtener una buena recomendación , que el usuario tuviera al menos dos artículos en común con algún otro usuario.

Vamos a explicar como sería la interacción entre el usuario “José” y la aplicación:

- Primeramente el usuario se registra en la aplicación.
- Nada más registrarse la aplicación le recomienda los cinco artículos siguientes:
  - Primer cata...
  - The volatile...
  - Aportación...
  - Floristic...
  - Tratamiento...

- El usuario podría consultar cualquiera de estos artículos recomendados y si así lo desea, incluirlo en su lista de favoritos con una puntuación.

Como se puede apreciar, al usuario no se le recomiendan artículos que ya tiene en su lista de favoritos. Tanto con "Jorge" como con "Cristina", "José" tiene dos artículos favoritos en común, sólo que "Jorge" tiene un perfil más parecido y por ello los tres primeros artículos recomendados provienen de él. Como la aplicación le recomienda cinco artículos y "Jorge" ya no tiene más artículos en favoritos, pasa a recomendarle artículos de "Cristina" que tiene un perfil bastante relacionado, ya que como se puede ver en la tabla de Facultades de la *figura 4.14,* las facultades de "Biología" y "Geología" tienen una similitud alta. "Cristina" tiene tres artículos en favoritos a parte de los que tiene en común con "José", y de ellos le recomienda los dos que tienen mayor puntuación, ya que a la hora de recomendar artículos de un usuario se eligen siempre en orden decreciente de puntuación.

Tras realizar más pruebas similares con el resto de usuarios que habíamos introducido y observando cual sería la recomendación más adecuada a cada usuario, comprobamos que el funcionamiento de la aplicación era el deseado.

4.2 Prototipo final

Una vez terminados los cuatro prototipos iniciales tuvimos una reunión con la coordinadora del proyecto Belén Díaz Agudo, el ayudante del proyecto Juan Antonio Recio García y un miembro de los servicios de la Biblioteca UCM, Laura Henche, para presentarles los prototipos. Tomamos nota de los errores cometidos, de las sugerencias por parte de la Biblioteca y de los próximos objetivos. Al final de dicha reunión decidimos que había que desarrollar un único sistema recomendador que incluyera las mejores características de cada uno de los prototipos iniciales.

Así pues, desarrollamos una única aplicación para recomendar artículos. Esta aplicación ofrece la posibilidad de realizar una recomendación desde dos puntos de vista diferentes: sin perfil de usuario y con perfil de usuario. En la primera opción, el sistema se comporta como un recomendador Conversacional, donde usuario y sistema interactúan varias veces hasta encontrar la solución deseada. Esta opción realiza la recuperación de casos tanto con metadatos del artículo (título, autor, materia principal…) como con el texto que forma el artículo (utilizando como función de similitud el algoritmo Lucene). Es por tanto, una mezcla de los prototipos explicados en los apartados *4.1.2 Single-Shot con Lucene* y *4.1.3 Conversacional*. En la opción con perfil de usuario el sistema se basa en comparar un perfil de usuario con el de otros usuarios y recomienda artículos que están marcados como favoritos en perfiles similares al del usuario activo. Es una ampliación del prototipo explicado en el apartado *4.1.4 Colaborativo.*

Ambos recomendadores están integrados dentro de la misma aplicación y simplemente cambiando de pestaña puedes usar uno u otro sistema de recomendación (*Figura 4.18*).

Figura 4.18

Es interesante comentar los diferentes enfoques de un recomendador con o sin perfil de usuario. En un sistema sin perfil la recomendación se basa en la similitud entre la consulta ylos casos que representan artículos. En un sistema con perfil la recomendación se basa en la similitud entre perfiles, y los artículos favoritos de perfiles similares son los que finalmente se recomiendan.

A continuación explicaremos por separado el funcionamiento de cada uno de estos sistemas recomendadores.

4.2.1 Conversacional con Lucene

Esta opción sigue el esquema de un recomendador conversacional, es decir, el usuario y el sistema interactúan varias veces hasta que el usuario decide que la recomendación ha terminado. El usuario rellena una consulta inicial, el sistema le propone unos candidatos, el usuario escoge un candidato y le añade una crítica y el sistema reformula la consulta con dicha información y le propone otro conjunto de candidatos y así cíclicamente.

Para la obtención de la consulta inicial (tarea *One-Off Preference Elicitation*) el sistema muestra un formulario que el usuario debe rellenar. El formulario está compuesto por una serie de campos que permiten describir un artículo. Estos campos son los mismos que en los prototipos anteriores (titulo, nombre del autor, apellidos del autor, materia principal, materia secundaria, palabras clave y revista) más dos campos nuevos (área y texto libre).

El campo área lo introdujimos como sugerencia de los servicios de la biblioteca y su inclusión nos permitió modificar aspectos del sistema que mejoraron el rendimiento del mismo. En total existen cuatro áreas que cubren todas las materias principales. Con este nuevo atributo nos pareció que sería interesante dividir la base de casos en cuatro conjuntos, según el área. Esta idea mejora los tiempos de recuperación de casos ya que solo se debe hacer similitud con el conjunto correspondiente al área que el usuario introduce en la consulta inicial. Esto tiene sentido pues si se está buscando un artículo de Humanidades no tiene mucho sentido que se haga similitud con artículos de Ciencias. A cambio, se añade la restricción de que siempre se debe rellenar el campo área, aunque no es un gran inconveniente. Las cuatro áreas son:

- Humanidades
- Ciencias
- Ciencias Sociales
- Ciencias de la Salud

El otro campo que añadimos al formulario es el de texto libre. El propósito de este campo es que el usuario escriba en lenguaje natural un resumen o cualquier información sobre el artículo que está buscando. Este texto se usa para compararlo con el texto de los artículos (las 50 primeras líneas) mediante el algoritmo de Lucene. Cuanto más se escriba en el campo texto libre más información habrá para hacer similitud con los textos de los artículos y mejores serán los resultados obtenidos.

En la *figura 4.19* se puede observar el formulario inicial que hemos comentado.

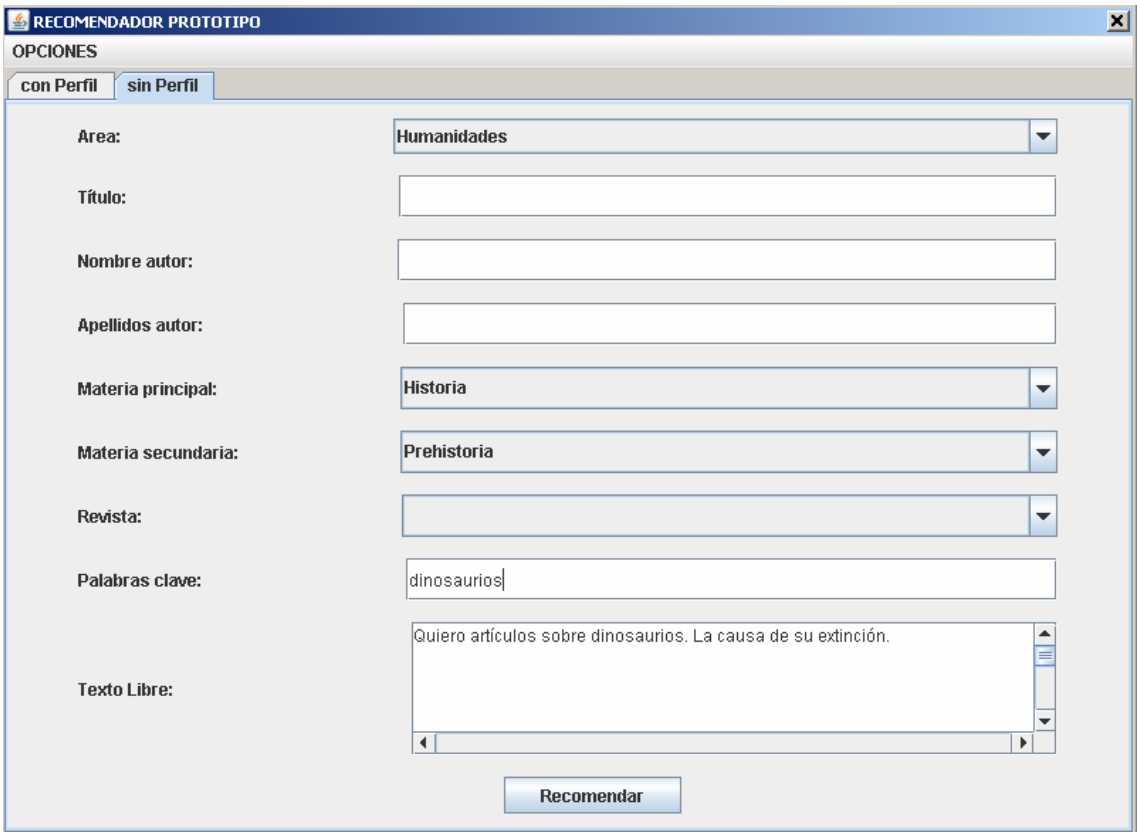


Figura 4.19

La estructura del programa principal es la misma que en casos anteriores: configurar base de casos y conector (método configure()), cargar la base de casos (método precycle()), ejecutar ciclo CBR (método cycle()) y cerrar el conector (postcycle()). Simplemente destacaremos los aspectos más novedosos de estos métodos.

- **Configure()**

El objetivo de este método es preparar al sistema para el proceso de recomendación. Se crea la base de casos lineal (*LinealCaseBase*), el conector a texto plano (*PlainTextConnector*) y el objeto que contendrá las funciones de similitud (*NNConfig*). Se escoge la estrategia que se va a utilizar como crítica (Something-Like-This). También se inicializa el conector a través de un xml que le dice de dónde coger la información para la base de casos y cómo interpretar dicha información.

Además, perfeccionamos un aspecto de los prototipos anteriores. Antes, para escoger una materia secundaria en la consulta se seleccionaba la pestaña correspondiente y aparecían todas las materias secundarias existentes (aproximadamente 750). Ahora, la idea es que si escoges una materia principal concreta, cuando vas a seleccionar una materia secundaria sólo aparezcan las materias secundarias que se engloban dentro de la misma materia principal. Para realizar esto debíamos leer de una base de datos de Access que contiene la relación entre cada materia principal y el conjunto de materias secundarias que le pertenecen. Así pues, para leer de esta base de datos se debe crear otro conector a bases de datos (*DataBaseConnector*) e inicializarlo desde un xml para que sepa de dónde obtener la información y cómo interpretarla.

- **Precycle()**

En el preciclo se debe cargar la base de casos. Aunque el sistema ya sabe donde están los metadatos de los artículos (porque el conector ha sido configurado correctamente) también es necesario saber la ruta absoluta a los textos de los artículos para la similitud con Lucene. No nos pareció conveniente hacer una ruta relativa a los ficheros pues probablemente esta no coincidiría con la ruta que tienen los servicios de la Biblioteca, y la integración sería más complicada. Así que antes de cargar la base de datos el usuario debe especificar la ruta absoluta a la jerarquía de carpetas que organiza los pdf's de los artículos (*Figura 4.20*).

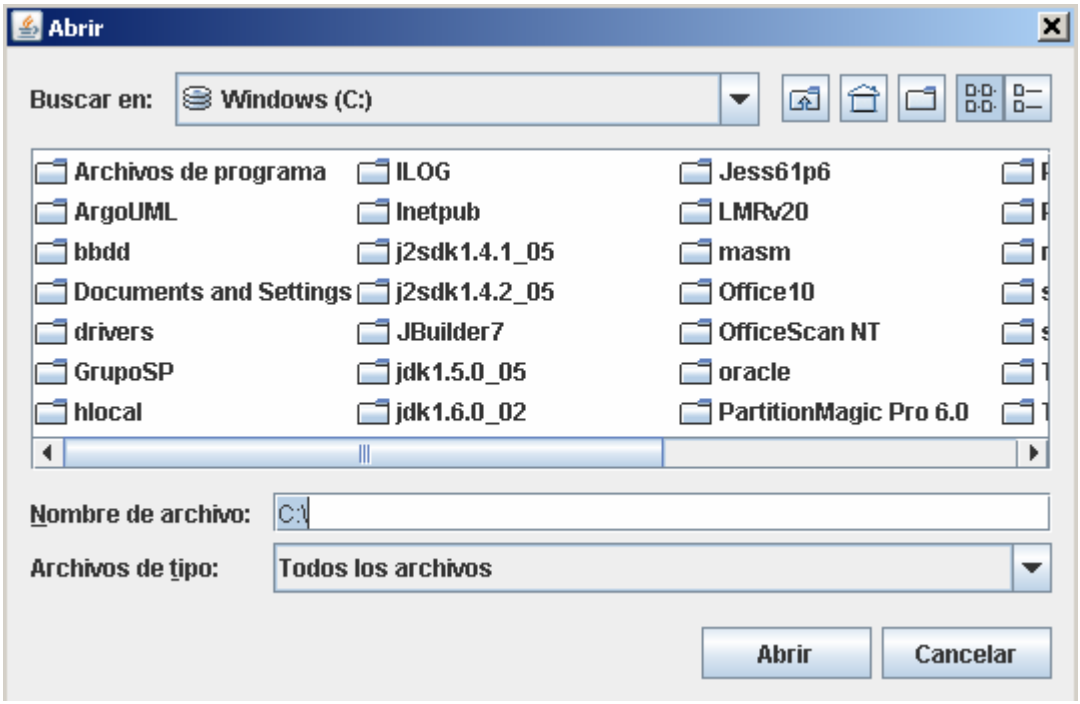


Figura 4.20

Una vez se conoce la ruta de los artículos se puede proceder a cargar la base de casos. Como se ha dicho anteriormente, por motivos de eficiencia decidimos definir cuatro bases de casos, una por cada área (Humanidades, Ciencias, Ciencias Sociales y Ciencias de la Salud). Como además de los metadatos hacen falta las 50 primeras líneas del artículo se debe acceder al propio texto del artículo. Esto se hace de la siguiente manera: cuando se leen los metadatos de un artículo uno de los campos es el nombre del pdf que contiene el texto (por ejemplo ARIS0000110071A.PDF). De este nombre se puede obtener la revista a la que pertenece y la ruta relativa en la jerarquía de directorios que organizan todos los artículos (ver nomenclatura de nombre de ficheros en el apartado *Proceso Catalogador 3.1*). Así se obtiene la ruta al pdf, se convierte el fichero pdf a un txt con la aplicación incorporada *pdftotext* y se leen las 50 primeras líneas que se mapean en el atributo contenido de la clase *Articulo_Description* de cada CBRCase.

Al igual que hay cuatro bases de casos para realizar comparación con Lucene hacen falta cuatro objetos *LuceneIndex* que contengan los textos normalizados de los artículos. Al final del preciclo tenemos las cuatro bases de casos cargadas y los cuatro índices de documentos Lucene, todo listo para realizar el ciclo CBR.

- **Cycle()**

El esquema que sigue este método es el de división en tareas de un recomendador conversacional. Primero se obtiene la consulta del usuario (*One-Off Preference Elicitation*) mediante un formulario (Form Filling). El aspecto del formulario es el que se ha mostrado anteriormente en la *figura 4.19*. Se mapean los campos del formulario en los atributos correspondientes de la descripción de la CBRQuery. Como ya se ha mencionado anteriormente el campo área es obligatorio rellenarlo y el campo texto libre se corresponde con el atributo *contenido* para compararlo con Lucene.

Una vez se tiene la query se procede a la recuperación de casos (*Retrieval*). Según el área de la query se comparará con la base de casos correspondiente. El método utilizado para la recuperación es el KNN que recupera los k-casos más similares. La función de similitud global es la media (*Average*) de las similitudes locales. Las funciones de similitud locales que utilizamos son: *TokensContained* para el titulo, autor, palabras clave y materia secundaria; *Equal* para la revista y el área; una tabla para la materia principal similar a la tabla de la *figura 4.14*; y Lucene para el campo texto libre. Para la comparación con Lucene, al existir cuatro índices de documentos uno por cada área, se utiliza el índice de documentos correspondiente al área de la query. Así pues, la similitud entre la consulta y la base de casos se realiza con el método *evaluateSimilarity* y se seleccionan los 10 casos más similares con el método *selectTopK*. Se muestran los casos recuperados en forma de tabla (*Figura 4.21*).

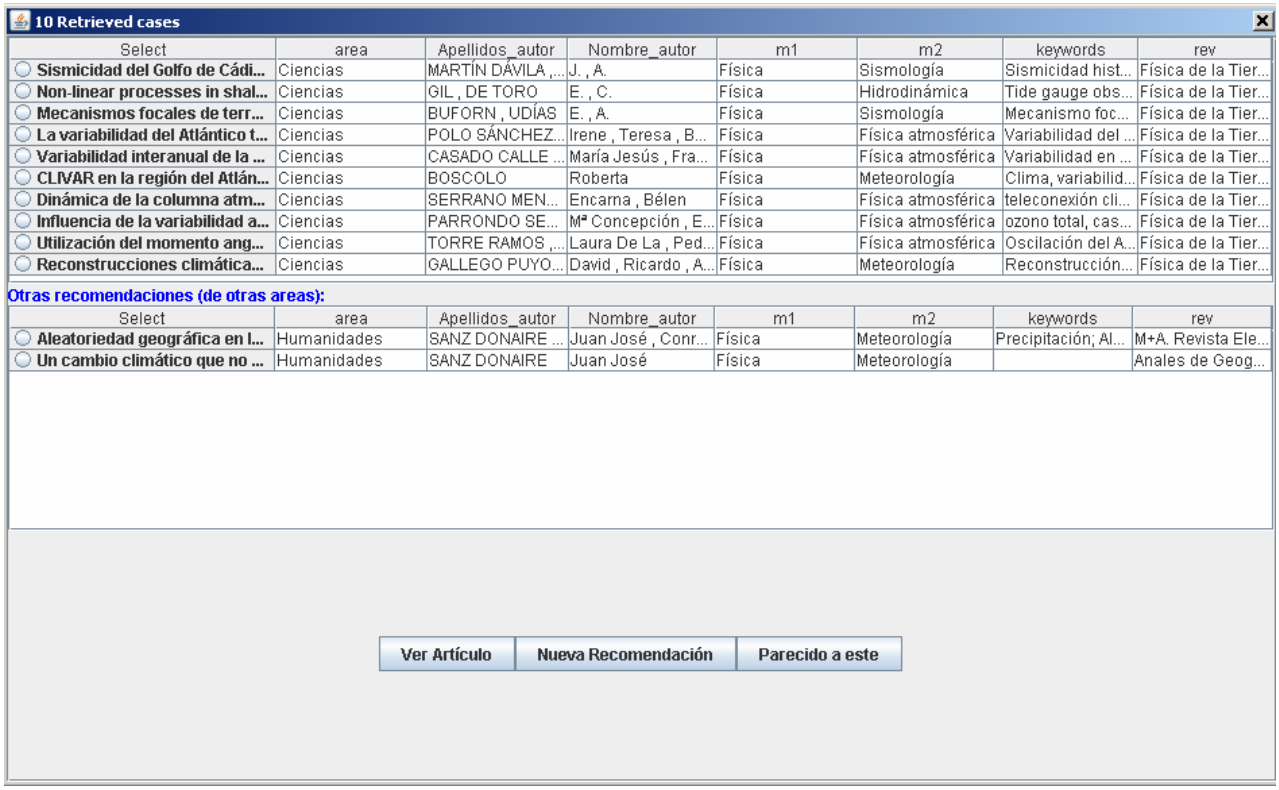


Figura 4.21

Como se puede observar en la *Figura 4.21* además de los 10 casos más similares del área que se introdujo en la consulta, también pueden aparecer casos recuperados de otras áreas distintas a la de la query. Decidimos añadir esta posibilidad ya que hay veces que las áreas se solapan en ciertas materias o el enfoque de una materia puede interpretarse en un área u otra. La manera en que se realiza la recuperación de casos de otras áreas es una variación del método KNN. En vez seleccionar los k-casos más similares, se seleccionan los casos con una similitud mayor que un cierto valor umbral. Después de realizar varias pruebas observamos que un valor umbral aceptable era 0.2, teniendo en cuenta que la evaluación tiene un rango entre 0 y 1.

Al ser un recomendador conversacional, al usuario se le ofrece la posibilidad de interactuar en varios ciclos con el sistema, seleccionando un caso y pidiendo al sistema un caso parecido (estrategia SomeThing-Like-This). En este caso el sistema combina la consulta anterior con el caso seleccionado. En general, para cada campo concatena la información de la consulta con la del caso seleccionado de manera que en la siguiente recuperación habrá más palabras en cada campo y la función de similitud *TokensContained* dará mejores resultados. Para la materia secundaria, no solo adjuntamos la materia secundaria del caso seleccionado sino todas las materias secundarias de la misma materia principal. El atributo contenido no se modifica, se mantiene el de la consulta inicial, a menos que no se hubiese rellenado el campo texto libre, en cuyo caso se copia el contenido de las 50 primeras líneas del caso seleccionado por el usuario.

Con la nueva consulta se vuelve a realizar la recuperación por KNN y se vuelven a mostrar los resultados. Este ciclo conversacional se repite hasta que el usuario encuentra el artículo que estaba buscando y se le ofrece la posibilidad de visualizarlo o abandonar la aplicación.

- **PostCycle()**

Cierra los conectores y termina la aplicación.

4.2.2 Colaborativo

Tras mostrar el Prototipo inicial, Laura Henche (Programadora del Dpto. de Automatización de la Biblioteca) nos propuso la mejora de que el usuario tenga la posibilidad de incluir varias Materias 2 y varios idiomas en su perfil de usuario. Nos pareció una buena idea ya que existe un conjunto muy amplio de Materias 2 (aproximadamente unas 750), y que el usuario sólo pueda elegir una de ellas como Materia 2 favorita es muy restrictivo.

Como comentamos anteriormente en el apartado 4.1.4 (Prototipo Colaborativo Inicial) no disponíamos de unas tablas con una estructura bien definida para cargar los casos en la Base de Conocimiento. Por ello, para este prototipo final, llegamos a un acuerdo con Laura para definir la estructura de las tablas, que nos permitirían cargar los casos incluyendo la mejora de que un usuario pueda tener varias Materias 2 y varios idiomas en su perfil de usuario. Se definieron dos tablas, cuyo esquema es el siguiente:

- Preferencias (id, email, campo, valor)
- Favoritos (id, email, favoritos)

En la tabla Preferencias se van a almacenar los perfiles de usuario, de forma que el campo “email” identifica unívocamente al usuario en cuestión, la columna campo indica que campo del

perfil se va a rellenar (Facultad, Materia 1, Materia 2 o Idioma), y por último, el campo "valor" se refiere al valor determinado para el campo elegido en la columna anterior. Por ejemplo, para un usuario perteneciente a la facultad de Biología tendríamos:

campo = 'Facultad' valor = 'Facultad de Ciencias Biológicas'

Como se puede observar en la definición de la tabla anterior, el perfil de un usuario va a estar almacenado en varias filas de la tabla Preferencias, y se permite la inclusión de varios valores para los campos Materia 2 e Idioma como era nuestra intención.

En la tabla Favoritos, para cada usuario identificado por el campo "email", se almacenarán en el campo "favoritos" la lista de sus artículos favoritos, siguiendo la siguiente codificación para cada artículo favorito en cuestión:

IdOrigen $ IdAjena $ Puntuación

- IdOrigen: Se refiere al recurso del cual proviene el documento. En nuestro caso siempre contendrá el valor "3", puesto que sólo recomendamos artículos del Portal de Revistas Electrónicas.

- IdAjena: Identifica unívocamente al documento dentro del recurso especificado en el IdOrigen. En nuestro caso, representa el nombre asignado al artículo.

- Puntuación: Es la puntuación asignada al documento. Es un valor comprendido entre "1" y "5".

Gracias a la inclusión del IdOrigen y del IdAjena, es fácilmente generalizable la recomendación de documentos de cualquier recurso de la Biblioteca, como ya comentamos anteriormente en el apartado 4.1.4.

Para un usuario cualquiera podríamos tener por ejemplo la siguiente lista de favoritos, en la que los distintos artículos están separados por medio de guiones:

3$BOCM0404110005A.PDF$3 – 3$BOCM0404110039A.PDF$5

Para poder relacionar estas dos tablas nos creamos la vista Perfil, cuyo esquema es el siguiente:

Perfil (id, email, campo, valor, favoritos)

A partir de ella, se realiza un mapeo para cargar en memoria los casos de la base de conocimiento. Los casos, como se comentó en el apartado 4.1.4 (Prototipo Colaborativo Inicial), están divididos en una Descripción, una Solución y un Resultado; y se cargarán en memoria de la siguiente manera:

- Descripción: Los distintos campos que forman el perfil de un usuario aparecen en varias filas de la vista Perfil. Para el mapeo en memoria se unifican todos ellos de forma que generan una única Descripción.

- Solución: Contiene un artículo favorito del usuario. Leemos una fila de la vista Perfil que se corresponda con el usuario en cuestión, y generamos una Solución por cada artículo que exista en el campo "favoritos" de ese usuario.

- Resultado: Almacena la puntuación asignada al artículo contenido en la componente Solución.

Por lo tanto, en la base de conocimiento tendrá tantos casos con la misma Descripción como artículos favoritos tenga el usuario al que corresponde la Descripción.

Para realizar la carga de los casos en la base de conocimiento se utiliza el conector de base de datos (DataBaseConnector) contenido en el framework jCOLIBRI2. Este conector nos permite leer los datos de la vista Perfil que serán procesados por el método *retrieveAllCases_Perfil* para crear los casos de la manera que hemos comentado.

Al introducir la mejora de que el usuario tenga la posibilidad de incluir varias Materias 2 y varios Idiomas en su perfil de usuario, cambiamos las funciones de similitud local que tenían estos campos del perfil en el Prototipo Inicial a la función de similitud local *TokensContained.* Esta función devuelve un valor comprendido entre "0" y "1" dependiendo del número de elementos en común que tengan los campos que se comparan.

4.2.2.1 Interfaz de la aplicación

Con la idea de que este Prototipo iba a ser explicado y mostrado en la Biblioteca, mejoramos la interfaz de la aplicación de forma que se pudiera visualizar el perfil de cada usuario y su lista de artículos favoritos, para facilitar así las pruebas que pudiera realizar posteriormente la Biblioteca en el testeo de la aplicación. En la *figura 4.22* se muestra la ventana principal de la aplicación, en ella se pueden elegir los diferentes usuarios y para cada uno de ellos visualizar su perfil. Mediante el botón "Ver favoritos" se pueden ver los artículos favoritos del usuario seleccionado. Para realizar la recomendación del usuario seleccionado se pulsa sobre el botón "Recomendar".

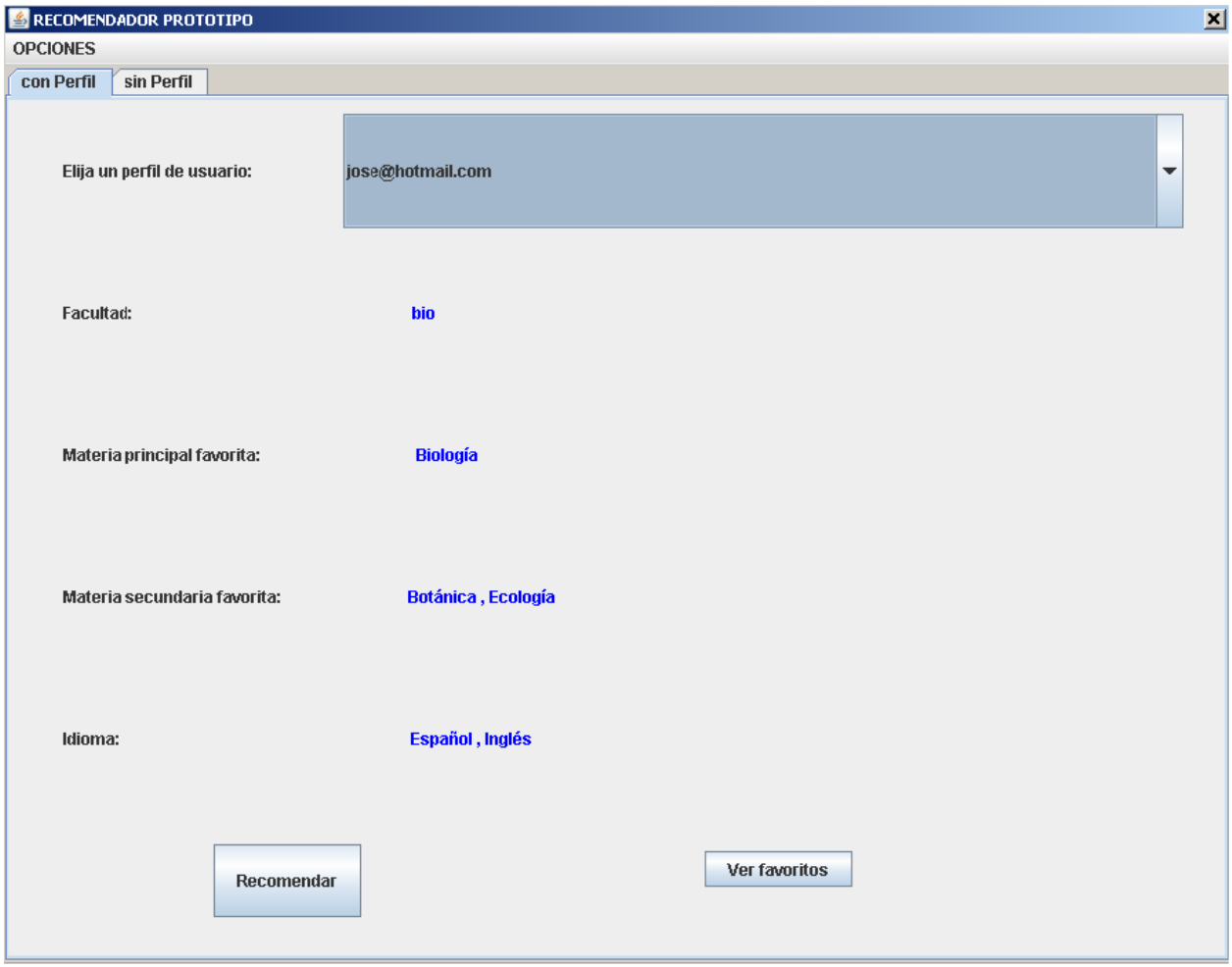


Figura 4.22

Si se ha elegido la opción "Ver favoritos", se visualiza la *figura 4.23* en la que se muestran los artículos favoritos del usuario activo. Cada artículo favorito tiene un número asignado (Número Favorito) que facilita la identificación de cada artículo en las recomendaciones.

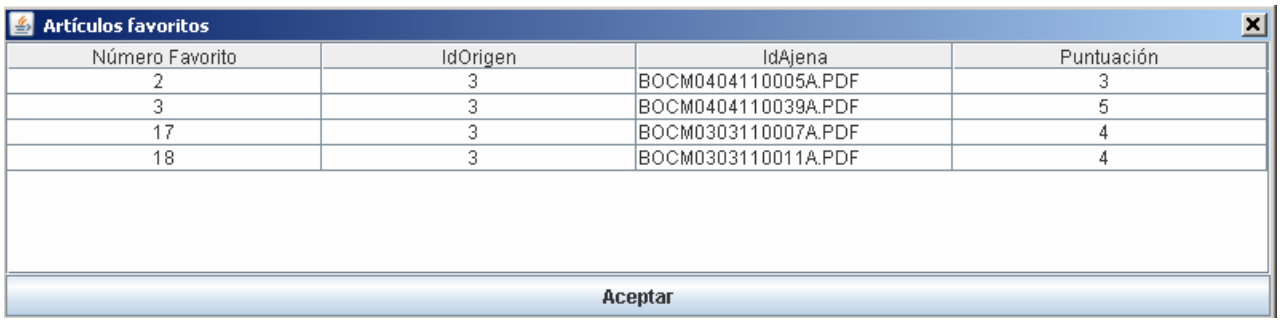


Figura 4.23

Si se ha seleccionado la opción "Recomendar", se mostrará la ventana de la *figura 4.24* en la que se pueden observar los cinco artículos recomendados. Cada artículo recomendado esta identificado por el "Número Favorito" comentado anteriormente, de forma que permite buscar fácilmente qué usuario o usuarios tienen ese artículo incluido en su lista de favoritos. Como se puede observar, no se recomiendan artículos que el usuario ya tenga incluidos en su lista de

artículos favoritos. Con la opción ”Ver Artículo” se puede ver el texto de cualquiera de los artículos recomendados que se haya seleccionado. Para volver a la ventana inicial de la aplicación (*figura 4.22*) se debe pulsar el botón “Nueva Recomendación”.

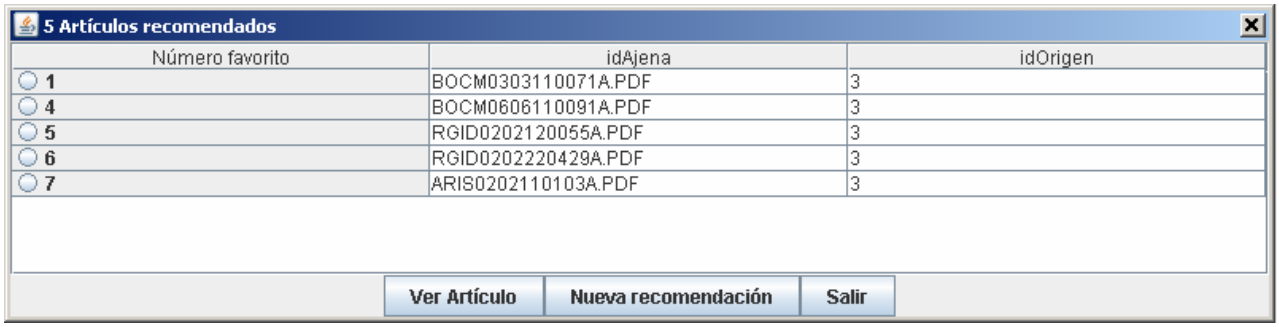


Figura 4.24

4.2.2.2 Experimentos

Debido a la inexistencia de datos reales en las tablas Preferencias y Favoritos, ya que es una aplicación nueva en la Biblioteca, incluimos en las tablas datos de diez usuarios. Cada uno de ellos tiene como artículos favoritos los siguientes:

- ana@hotmail.com : 1, 2, 3, 4
- jose@hotmail.com: 2, 3, 17, 18
- jorge@hotmail.com: 3, 4, 14, 15, 16
- jesus@hotmail.com: 7, 8, 9, 10, 11, 12, 13
- cristina@hotmail.com: 5, 6
- maite@hotmail.com: 23, 24, 25, 26, 27, 28
- roberto@hotmail.com: 7, 11, 12, 20, 21, 25, 26, 30, 31
- pablo@hotmail.com: 5, 6, 9, 10, 13
- maria@hotmail.com: 9, 10, 22, 29
- juan@hotmail.com: 9, 19, 20, 21, 22

En la siguiente tabla (*Figura 4.25*) se muestran las similitudes entre perfiles de usuarios (valores comprendidos entre “0” y “1”):

	José	Ana	Roberto	Juan	María	Jorge	Jesús	Cristina	Pablo	Maite
José	X	0.85	0.38	0.21	0.21	0.55	0.35	0.53	0.58	0.48
Ana		X	0.35	0.35	0.35	0.7	0.35	0.55	0.55	0.45
Roberto			X	0.55	0.54	0.33	0.85	0.33	0.34	0.6
Juan				X	0.87	0.16	0.43	0.2	0.34	0.54
María					X	0.16	0.53	0.16	0.45	0.28
Jorge						X	0.35	0.33	0.43	0.43
Jesús							X	0.33	0.55	0.55
Cristina								X	0.89	0.33
Pablo									X	0.35
Maite										X

Figura 4.25

Como ejemplo ilustrativo, vamos a realizar una recomendación para el usuario "Roberto" cuyo perfil es el siguiente:

- Facultad: Facultad de Bellas Artes
- Materia 1: Bellas Artes
- Materias 2: Educación Artística, Dibujo, Logopedia, Psicología Experimental
- Idiomas: Español, Inglés

Los artículos que se le recomiendan al usuario "Roberto" son los mostrados en la *Figura 4.26*:

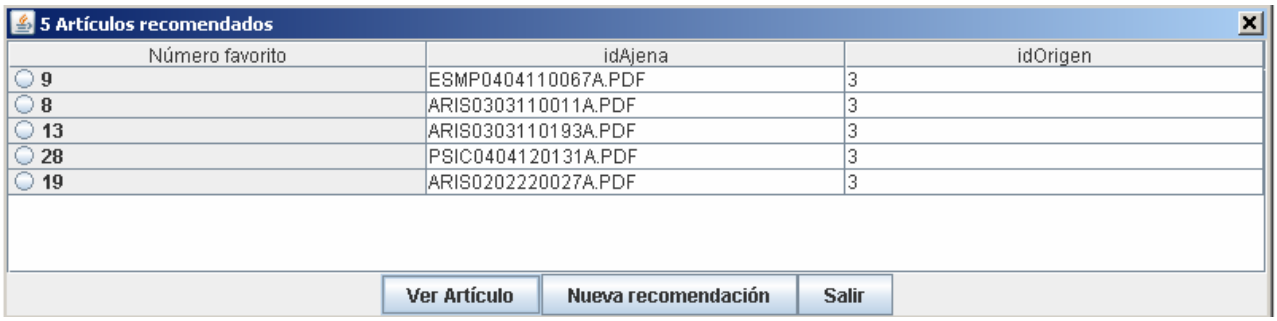


Figura 4.26

Como se puede observar en la *figura 4.26,* los tres primeros artículos recomendados tienen como "Número Favorito": 9, 8, 13. Estos artículos provienen de la lista de artículos favoritos del usuario "Jesús". Las razones por la que se le recomiendan artículos de "Jesús" son las siguientes:

- "Jesús" es el usuario con el que más artículos tiene en común "Roberto", en concreto tienen tres artículos en común.
- Como se puede observar en la *figura 4.25*, "Jesús" es el usuario con el que tiene mayor similitud de perfil. Este valor de similitud tan alto se debe a que ambos usuarios pertenecen a la misma Facultad, tienen la misma Materia 1 favorita y también tienen gustos parecidos en las Materias 2.

Por estas razones, resulta lógico que a "Roberto" le vayan a interesar los artículos favoritos de "Jesús", y por eso la aplicación le recomienda primeramente estos artículos.

El artículo con "Número Favorito" 28 proviene de la lista de artículos favoritos del usuario "Maite". Esto se debe a que los usuarios "Roberto" y "Maite" tienen dos artículos favoritos en común. Además, "Maite" pertenece a la Facultad de Psicología y "Roberto" está muy interesado en ese tema, puesto que tiene varias Materias 2 favoritas de psicología que también tiene "Maite".

Por último, el artículo con "Número Favorito" 19 proviene de la lista de artículos favoritos del usuario "Juan". La razón por la que se recomienda este artículo es que este usuario es el único, a parte de "Jesús" y "Maite", que tiene más de un artículo en común con "Roberto".

Como se puede observar, primero se han recomendado los artículos de “Jesús” y de “Maite”, ya que aparte de tener artículos en común con Roberto también tienen un perfil más parecido que “Juan”.

Realizamos más pruebas similares con el resto de usuarios, comprobando que los resultados obtenidos eran los deseados y que el funcionamiento de la aplicación era el correcto.

5. INTEGRACIÓN CON LA BIBLIOTECA

Como última fase del desarrollo se procedió a la presentación de los sistemas descritos en los apartados 3 y 4. En una reunión en el edificio de la biblioteca de la UCM, les explicamos su funcionamiento, les proporcionamos los ejecutables y un manual de usuario para que pudieran probarlos. Ellos se reunieron para ponerse de acuerdo sobre cómo aplicar y darle utilidad a las aplicaciones que les habíamos presentado, y sobre todo decidir si era viable su integración.

Como conclusiones podemos destacar las siguientes: al recomendador le veían muchas posibles utilidades, desde incluirlo en el Campus Virtual a utilizarlo como recomendador con perfil de usuario, o recomendador sin perfil... Esto unido al hecho de que era necesario mejorar la eficiencia en tiempo del recomendador presentado, les hizo tomar la decisión de dejar para más adelante su implantación y tener así tiempo suficiente para decidir la mejor manera de aplicarlo a sus necesidades.

En cuanto al catalogador, vieron muy útil y necesario su implantación inmediata, por lo que nos reunimos con un experto en el dominio, Laura Henche (Programadora del Dpto. de Automatización de la Biblioteca), para que nos explicara cómo quería la Biblioteca que se realizara la integración y cuales eran sus requisitos.

En el siguiente apartado vamos a profundizar en la integración del catalogador en la Biblioteca.

5.1 Integración del catalogador

Durante la reunión que tuvimos con el experto nos comentó que les gustaba más la idea de utilizar el catalogador como un “servicio”, es decir, que ellos generan un fichero de texto con los datos de los artículos que se quieren catalogar, se le pasa a la aplicación, y esta devuelve otro fichero de texto con los resultados obtenidos. Todo este proceso se realizaría por consola mediante comandos, por lo que la interfaz gráfica ya no sería necesaria. Eliminando la interfaz gráfica se simplifica mucho el proceso de integración puesto que ya no es necesaria la adaptación a una aplicación Web.

En el siguiente subapartado vamos a profundizar en las peticiones que nos propuso sobre la integración del catalogador.

5.1.1 Peticiones

La idea que tenían en la Biblioteca era poder utilizar el catalogador en dos modos:

- Integrado en la catalogación diaria: Formaría parte del proceso que realizan los bibliotecarios en la catalogación los artículos. De la misma forma que comentamos en el apartado 3.1 de Adquisición de conocimiento, lo primero que harían los bibliotecarios sería formar el nombre del artículo, y realizar una copia del fichero ".PDF" en el servidor.

  Lo siguiente sería generar un fichero de texto con las rutas absolutas y los nombres de los artículos que se van a catalogar. Este fichero de texto se le pasa como parámetro de entrada al catalogador, que devolverá en otro fichero el resultado de la catalogación y el tanto por ciento de seguridad con el que se han obtenido los resultados.

  Este fichero resultado se le presenta al bibliotecario para que revise la información extraída, una vez revisada se escribe en otro fichero de texto que será enviado de vuelta al catalogador para que éste aprenda y amplíe su base de conocimiento.

- Para la catalogación del fondo histórico: El histórico está formado por un conjunto de aproximadamente 25.000 artículos sin catalogar. La idea es realizar una catalogación masiva de estos artículos.

  En este modo se incluyen dos parámetros de entrada a la aplicación, el primero es un fichero similar al del modo de catalogación diaria, el segundo es opcional y representa el tanto por ciento de seguridad a partir del cual se va a considerar segura la catalogación de un artículo.

  El resultado de la aplicación va a ser devuelto en dos ficheros. En un fichero se incluyen los artículos catalogados con una seguridad mayor o igual a la especificada en el parámetro de entrada, los cuales se cargarán directamente sin pasar previamente por la revisión de los bibliotecarios. En el otro fichero se incluyen los artículos dudosos, cuya seguridad es inferior a la especificada en el parámetro de entrada, y los cuales serán revisados por el bibliotecario antes de ser cargados. Tras la revisión, igual que en el modo de catalogación diaria, el fichero generado será enviado de vuelta a la aplicación para mejorar su aprendizaje.

Además, el experto también nos comentó que sería de gran utilidad la extracción de más metadatos del artículo como el *Título, Clase, Autores*... A parte de estos metadatos que ya conocíamos, nos propuso la extracción de dos nuevos campos:

- Idioma 1: representa el idioma principal del artículo.
- Idioma 2: representa el idioma secundario del artículo.

Un artículo en español que contiene *Resumen y Abstract,* incluirá en el texto primero el *Resumen* y después el *Abstract,* por lo que tendrá como *Idioma 1* el español y como *Idioma 2* el inglés. Si el artículo está en inglés e incluye los campos *Resumen y Abstract,* contendrá en primer lugar el *Abstract,* por lo que como *Idioma 1* quedará el inglés y como *Idioma 2* el español. En caso de que el artículo sólo contenga *Resumen o Abstract* el campo *Idioma 2* quedará vacío, y si no incluye ni *Resumen* ni *Abstract* también quedará vacío el campo *Idioma1*.

Como última proposición, nos comentó que les gustaría que la aplicación tuviese un uso más general, de tal forma que se pudiera catalogar cualquier documento como por ejemplo las tesis, complured...

Del estudio de las distintas posibilidades, vimos que era viable la realización de los dos modos de funcionamiento y la extracción de los campos *Idioma 1 e Idioma 2*. En cambio, la extracción de otros metadatos como *Título o Autores* no era viable, ya que como comentamos anteriormente en el apartado 3.2 (Desarrollo del Catalogador), los artículos no siguen ninguna estructura para estos campos por lo que no podemos utilizar expresiones regulares para extraerlos. En cuanto a la idea de generalizar el uso del catalogador, decidimos posponerlo y centrarnos en finalizar correctamente el catalogador de artículos.

En el siguiente apartado profundizamos en el desarrollo de cada parte del nuevo catalogador.

5.1.2 Desarrollo

El esquema para el modo de la catalogación diaria mostrado en la *figura 5.1* sería el siguiente:

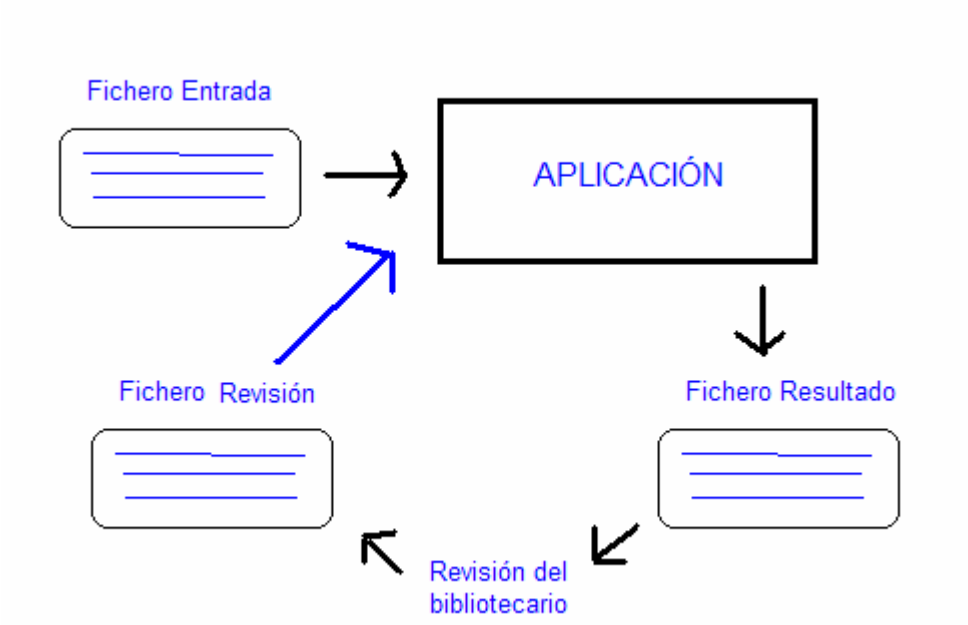


Figura 5.1

Una vez definido el esquema que íbamos a seguir, el siguiente paso era definir la estructura de los ficheros con los que íbamos a trabajar.

- Fichero Entrada: Está formado por la ruta absoluta y el nombre de los artículos a catalogar. En cada línea del fichero se pone la información de un artículo de la siguiente forma:

  Ruta # Nombre del fichero

- Fichero Revisión: A partir de este fichero se incluyen nuevos casos en la base de conocimiento. Se genera una vez que el bibliotecario ha revisado el fichero resultado.

Está formado por el nombre del artículo y su Materia 2 correspondiente, ya que es el único metadato necesario para formar los casos que se incluyen en la base de conocimiento. En cada línea del fichero se pone la información de un artículo de la siguiente forma:

Nombre del artículo # Materia 2

En el caso de que un artículo tenga varias materias 2, en el "Fichero Revisión" habrá una línea por cada materia 2 que tenga.

- Fichero Resultado: Está formado por la información obtenida como resultado de la catalogación de cada uno de los artículos incluidos en el "Fichero Entrada". En cada línea del fichero se pone la información de un artículo de la siguiente forma:

Nombre || Materia2 || Materia1 || Resumen || P.Clave || Abstract || Keywords || Idioma1 || Idioma2 || % Seguridad

Si durante la catalogación no se ha podido extraer alguno de los metadatos, su campo correspondiente se deja vacío.

El nuevo campo *Seguridad* representa el tanto por ciento de seguridad con el que se ha obtenido el resultado para los campos *Materia 2 y Materia 1* en la catalogación del artículo. Para calcular el valor de este campo realizamos varias pruebas con artículos de los cuales ya conocíamos su *Materia 2* y su *Materia 1*. En estas pruebas nos fijamos principalmente en el campo *Materia 1* de cada artículo, ya que un artículo sólo tiene una *Materia 1* pero puede tener varias *Materias 2*, por lo que a la hora de calcular la seguridad vimos más significativa la *Materia 1*.

Como ya comentamos anteriormente, para cada artículo a catalogar se recuperan los cinco artículos más parecidos de la base de conocimiento según la medida de similitud descrita en el apartado 3.2.1, pero de esos cinco artículos recuperados sólo se tienen en cuenta aquellos con un grado de similitud mayor a 0.5 ("artículos válidos"). Para calcular la *Seguridad* se utiliza un sistema de votación, de forma que sólo se tiene en cuenta el voto de los "artículos válidos". Cada "artículo válido" vota por una *Materia 1*.

Observamos los resultados obtenidos en las pruebas realizadas, teniendo en cuenta para cada caso el número de "artículo válidos" y el número de *Materias 1* distintas votadas, y de esta forma obtuvimos un conjunto de reglas que utilizamos para el cálculo del campo *Seguridad.*

El caso mejor se da si todos los "artículos válidos" votan por la misma *Materia1*, en este caso se le asigna al artículo un 100% de seguridad. El caso peor se da si hay cinco "artículos válidos" y cada uno vota por una *Materia 1* distinta, entonces se le asigna al artículo un 20% de seguridad. Generamos reglas para todas las combinaciones posibles que se podían dar, teniendo en cuenta el número de "artículos válidos" y el número de *Materias 1* diferentes votadas por ellos.

Sólo se asignará 0% de seguridad a aquellos artículos que estén escaneados o protegidos, y cuyos textos no se pueden procesar para el cálculo de la similitud.

Una vez definida la estructura del modo de catalogación diaria, pasamos a especificar el esquema para el modo de la catalogación del fondo histórico que se muestra en la *figura 5.2:*

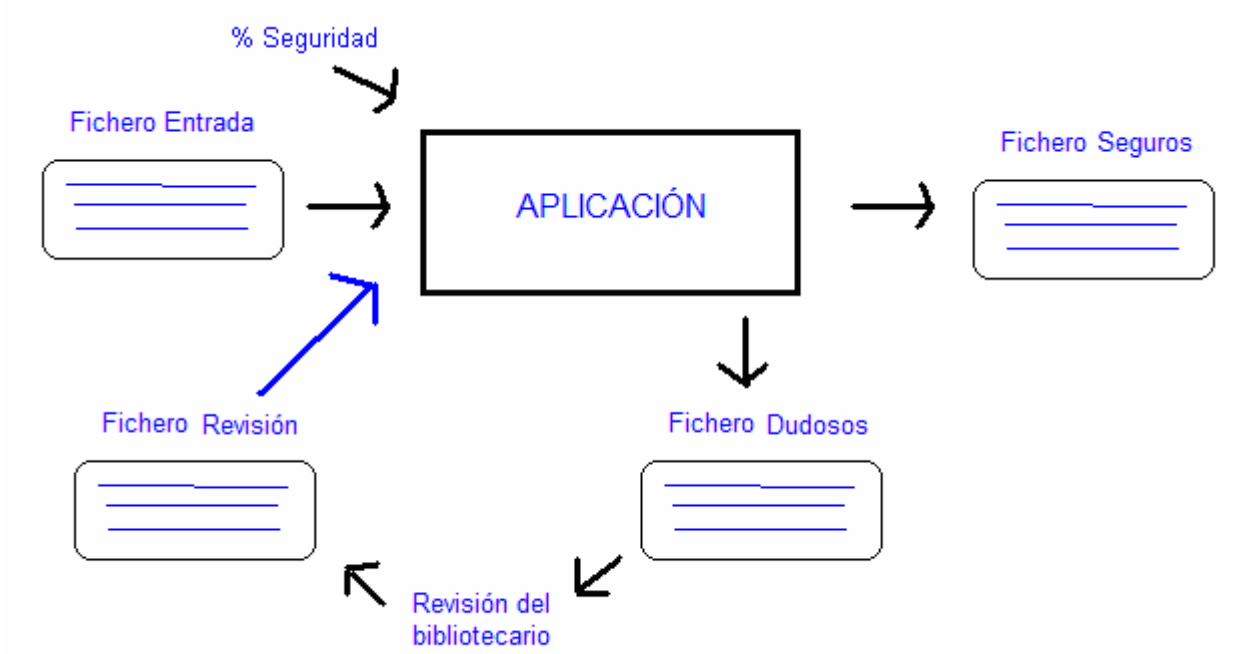


Figura 5.2

Como se puede observar en la *figura 5.2,* el esquema para la catalogación del fondo histórico es similar al esquema para la catalogación diaria de la *figura 5.1.* Sólo hay dos diferencias, la primera es que en este modo se añade un parámetro de entrada más, que es opcional y representa el tanto por ciento de seguridad a partir del cual se va a considerar segura la catalogación de un artículo. La segunda es que se generan dos ficheros de salida, ambos con la misma estructura que el "Fichero Resultado" de la *figura 5.1*:

- Fichero Seguros: Está formado por el resultado de todos los artículos incluidos en el "Fichero Entrada". La única particularidad es que aquellos artículos cuyo resultado se ha obtenido con una seguridad menor al parámetro de entrada "% Seguridad", tienen en su resultado los campos *Materia 1 y Materia 2* vacíos. Los artículos incluidos en este fichero se cargarán directamente sin pasar por la revisión del bibliotecario.

- Fichero Dudosos: Es el fichero que se va a presentar al bibliotecario para su revisión y está formado por el resultado de aquellos artículos cuya seguridad es menor a la especificada en el parámetro de entrada "% Seguridad".

Como ya hemos comentado, el parámetro de entrada "% Seguridad" es opcional y en el caso de que no esté especificado la aplicación le asigna un valor del 100%.

La idea de incluir los artículos cuya seguridad sea menor a la especificada en el parámetro de entrada en los dos ficheros resultado, viene dada por el hecho de que el fondo histórico está formado por un conjunto muy amplio de artículos sin catalogar y en la Biblioteca están interesados en obtener metadatos de estos artículos cuanto antes. Por ello, al incluirlos en el "Fichero Seguros", aunque tengan vacíos los campos de *Materia1 y Materia2,* la Biblioteca considera suficiente que tengan rellenos el resto de metadatos para cargar los artículos como ya catalogados

en su servidor. Como también están incluidos en el "Fichero Dudosos", una vez que el bibliotecario los revise se actualizarán en el servidor, incluyendo los campos *Materia1 y Materia2*.

De los artículos incluidos en el "Fichero Seguros", sólo aquellos cuya seguridad sea del 100% son incluidos directamente en la base de conocimiento, el resto aunque tengan una seguridad mayor o igual a la especificada en el parámetro de entrada no se incluyen en la base de conocimiento, ya que puede incluir entre sus *Materias 2* asignadas alguna que no sea correcta, y si se incluye en la base de conocimiento puede provocar que la aplicación cometa posteriormente errores puesto que no estará teniendo un buen aprendizaje.

Una vez definimos los esquemas de los dos modos de catalogación y la estructura de los diferentes ficheros, nos dimos cuenta de que nuestra aplicación iba a realizar tres operaciones claramente diferenciadas, catalogación diaria, catalogación del histórico y procesamiento del fichero de revisión para el aprendizaje. Por ello, pensamos que la mejor manera para diferenciar estas tres operaciones o modos, era incluir un nuevo parámetro de entrada de forma que:

- -c : para la catalogación diaria
- -b : para la catalogación del fondo histórico
- -r : para procesar el fichero de revisión e incluir así nuevos casos en la base de conocimiento.

Sólo nos faltaba decidir el momento en el que ejecutar el mantenimiento, necesario ya que la base de conocimiento va a ir creciendo aumentando así la precisión, pero llegará un punto en el que se verá afectada la eficiencia puesto que al tener una base de conocimiento muy amplia la aplicación tardará más tiempo en catalogar los artículos. Pensamos en avisar por consola al usuario en el momento que fuera necesario realizar el mantenimiento y añadir una nueva opción de entrada para ejecutarlo.

Comentamos esta opción al experto, pero pensó que sería más cómodo para ellos que lo ejecutáramos de manera automática. Nos comentó que pensaban ejecutar por la noche la catalogación del fondo histórico y el procesamiento de las revisiones, para así no ralentizar durante el día el trabajo de la catalogación diaria de los bibliotecarios. Por ello, tomamos la decisión de incluir el mantenimiento en el modo de catalogación del fondo histórico (-b) y en el modo de revisión (-r). De esta forma, una vez que se haya terminado de ejecutar la revisión se comprueba si el tamaño de la base de conocimiento supera un determinado umbral y si es así se ejecutará el mantenimiento. De forma análoga ocurrirá con la catalogación del fondo histórico.

Para determinar el umbral a partir del cual se debe ejecutar el mantenimiento, decidimos que como la base de conocimiento representativa inicial está formada por 412 artículos sería una buena idea fijar el valor del umbral al doble de esta cifra.

El experto nos comentó que pensaban dividir la catalogación de los artículos del fondo histórico en varias tandas. Quería saber de qué forma se iban a obtener los mejores resultados, si catalogando una única tanda cada noche, o varias de menor tamaño, cual sería el tamaño ideal para las tandas y si sería útil catalogar dos veces una misma tanda. Nosotros basándonos en las pruebas que habíamos realizado, le aconsejamos que el tamaño de las tandas fuera de 1000 artículos como máximo, y que ejecutaran varias tandas de artículos cada noche puesto que durante la ejecución de cada tanda se incluyen nuevos artículos en la base de conocimiento, que pueden ser útiles para obtener mejores resultados en tandas posteriores. Por esta misma razón, también le aconsejamos que sería mejor catalogar dos veces la misma tanda de artículos.

Finalmente, la aplicación va a ser llamada de tres maneras diferentes:

- catalogador.bat –c FicheroEntrada → Catalogación Diaria
- catalogador.bat –b FicheroEntrada [%Seguridad] → Catalogación del histórico
- catalogador.bat –r FicheroRevisión → Procesamiento FicheroRevisión y aprendizaje

Por último, sólo nos faltaba ponernos de acuerdo con Laura sobre los nombres que se le iban a asignar a los diferentes ficheros. Ella estaba interesada en que el nombre de los ficheros incluyera la fecha para poder mantener un orden. Por ello, decidimos que el nombre del "Fichero Entrada" que generan en la Biblioteca incluyera la fecha del día en el que se ha mandado su catalogación, y luego a partir de ese nombre generar el nombre de los ficheros de salida. De forma que, el nombre del fichero de salida de la catalogación diaria será el nombre del fichero de entrada concatenado con la palabra "Salida", y de la misma forma, para los nombres de los ficheros de salida de la catalogación del histórico, concatenaremos las palabras "SalidaDudosos" y "SalidaSeguros" al nombre del fichero de entrada. De esta manera, se van a poder distinguir los ficheros de salida de los de entrada, ya que los ficheros de salida van a generarse en el mismo directorio en el que estén los ficheros de entrada.

Una vez finalizada la aplicación, la llevamos a la Biblioteca para implantarla y explicarles su funcionamiento. Durante la implantación el problema principal fue que en la máquina en la que iba a ser implantada la aplicación disponía de un sistema operativo de tipo Unix, por lo que tuvimos que adaptar nuestra aplicación puesto que estaba desarrollada en Windows.

6. CONCLUSIONES Y LÍNEAS FUTURAS

Durante la realización de este proyecto hemos aprendido a desarrollar sistemas CBR (Razonamiento basado en casos). Este tipo de sistemas tiene un uso muy amplio, y son especialmente adecuados en dominios donde el aprendizaje juega un papel muy importante como en el caso del catalogador automático de textos. En esta aplicación es muy importante la fase de aprendizaje en la que se incluyen nuevos casos en la base de conocimiento, gracias a lo cual se obtienen mejores resultados en la catalogación de los documentos. Los nuevos casos introducidos en la base de conocimiento pasan previamente por una fase de revisión realizada por los bibliotecarios, esta fase garantiza el buen aprendizaje de la aplicación.

El uso de CBR en el catalogador nos facilitó también la obtención del resultado, ya que es mucho más rápido poder reutilizar soluciones previas dadas a casos similares que obtener las soluciones desde cero.

El catalogador está únicamente centrado en la catalogación de artículos del Portal de Revistas Electrónicas de la UCM, pero en un futuro se podría ampliar fácilmente para realizar la catalogación de otros recursos de la biblioteca como las tesis o complured. Simplemente habría que analizar las estructuras de estos otros recursos y a partir de ellas estudiar qué metadatos se podrían extraer en cada caso.

Existe una gran diversidad de sistemas recomendadores. Nosotros con la ayuda de las plantillas de recomendadores ([6]) desarrollamos varios prototipos iniciales, para luego aplicar lo mejor de cada uno de ellos en la creación de un único recomendador que cubriera todas las necesidades que se pudieran presentar en la biblioteca. El uso de nuestro recomendador está orientado a los usuarios de la biblioteca, y en concreto está centrado en la recomendación de artículos del Portal de Revistas Electrónicas de la UCM. Al igual que en el caso del catalogador, se podría ampliar el recomendador de forma que pudiera recomendar otro tipo de recursos o documentos existentes en la biblioteca, o desarrollar fácilmente con la ayuda de las plantillas otros recomendadores centrados en otro tipo de recursos de la biblioteca, o incluso orientados a otro tipo de usuarios, ya que nuestro recomendador utiliza un perfil de usuario pensado para profesores y alumnos.

Debido al amplio abanico de posibilidades en el desarrollo del recomendador, en la biblioteca no fueron capaces de tomar una decisión sobre el tipo de recomendador que les interesaba más ni de donde implantarlo, puesto que no se esperaban un tipo de aplicación como la que les presentamos. Por ello, decidieron posponerlo para tener tiempo suficiente para pensar de qué manera les gustaría aplicar este tipo de sistemas. Nuestro recomendador fue para ellos el inicio de un proyecto que no imaginaban que pudiera llevarse a cabo.

Como conclusión final, comentar que hemos realizado dos de las tres líneas de posible colaboración planteadas como objetivos iniciales del proyecto, una de ellas la de catalogación automática de textos ha podido ser implantada para el uso diario de la biblioteca, facilitándoles además la catalogación de un fondo histórico de aproximadamente 25.000 artículos que se encontraban sin catalogar, y que sin la ayuda de este proceso automático les hubiera llevado mucho tiempo. En cuanto a la segunda línea de colaboración, la del desarrollo de un recomendador, no ha sido posible su implantación en la biblioteca, pero ha servido para que la biblioteca pueda ver los servicios que ofrecen los sistemas recomendadores y así poder pensar en un futuro donde pueden ofrecer este tipo de servicios.

7. BIBLIOGRAFÍA

[1.] Derek Bridge, Mehmet H. Göker, Lorraine McGinty and Barry Smyth (2006): "Case-based recommender systems". *The Knowledge Engineering Review, Vol. 20:3, 315-320.*

[2.] M. Lenz. "Defining knowledge layers for textual case-based reasoning" *Proceedings of the 4th European Workshop on Advances in Case-Based Reasoning, EWCBR-98, pages 298-309, Springer-Verlag, 1998.*

[3.] Ramón López de Mantaras, David Mcsherry, Derek Bridge , David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary Lou Maher, Michael T. Cox, Kenneth Forbus, Mark Keane, Agnar Aamodt and Ian Watson. (2006): "Retrieval, reuse, revision and retention in case-based reasonig". *The Knowledge Engineering Review, Vol. 20:3, 215-240.*

[4.] Juan A. Recio-García, Belén Díaz Agudo y Pedro González Calero (Enero 2008): "Tutorial jCOLIBRI2". Technical Report IT/2007/02, Dep. Ingeniería de Software e Inteligencia Artificial, Universidad Complutense de Madrid, España.

[5.] Juan A. Recio-García, Belén Díaz-Agudo, Marco A. Gómez-Martín, and Nirmalie Wiratunga. (August 2005) "Extending jCOLIBRI for Textual CBR." *Proceedings of Case-Based Reasoning Research and Development, 6th International Conference on Case-Based Reasoning*, ICCBR 2005, pages 421-435, Chicago, IL, US, Springer.

[6.] Juan A. Recio-García, Derek Bridge, Belén Díaz-Agudo, and Pedro A. González-Calero. "CBR for CBR: A Case-Based Template Recommender System for Building Case-Based Systems". *European Conference on Case Based Reasoning*, ECCBR-2008. Springer-Verlag.

[7.] Juan A. Recio-García, Belén Díaz-Agudo and Pedro A. González-Calero. "Textual CBR in jCOLIBRI: From Retrieval to Reuse".

[8.] Juan A. Recio-García, Belén Díaz-Agudo, Derek Bridge and Pedro A. González-Calero. "Semantic Templates for Designing Recommender Systems".

[9.] C.J. van Rijsbergen, S.E. Robertson and M.F. Porter (1980). "New models in probabilistic information retrieval" *London, British Library Research and Development Report, no. 5587).*