2017/2018

Universidad Zaragoza

Diseño del Sistema de Recuperación

RECUPERACIÓN DE INFORMACIÓN

DANIEL MARTÍN – 702858

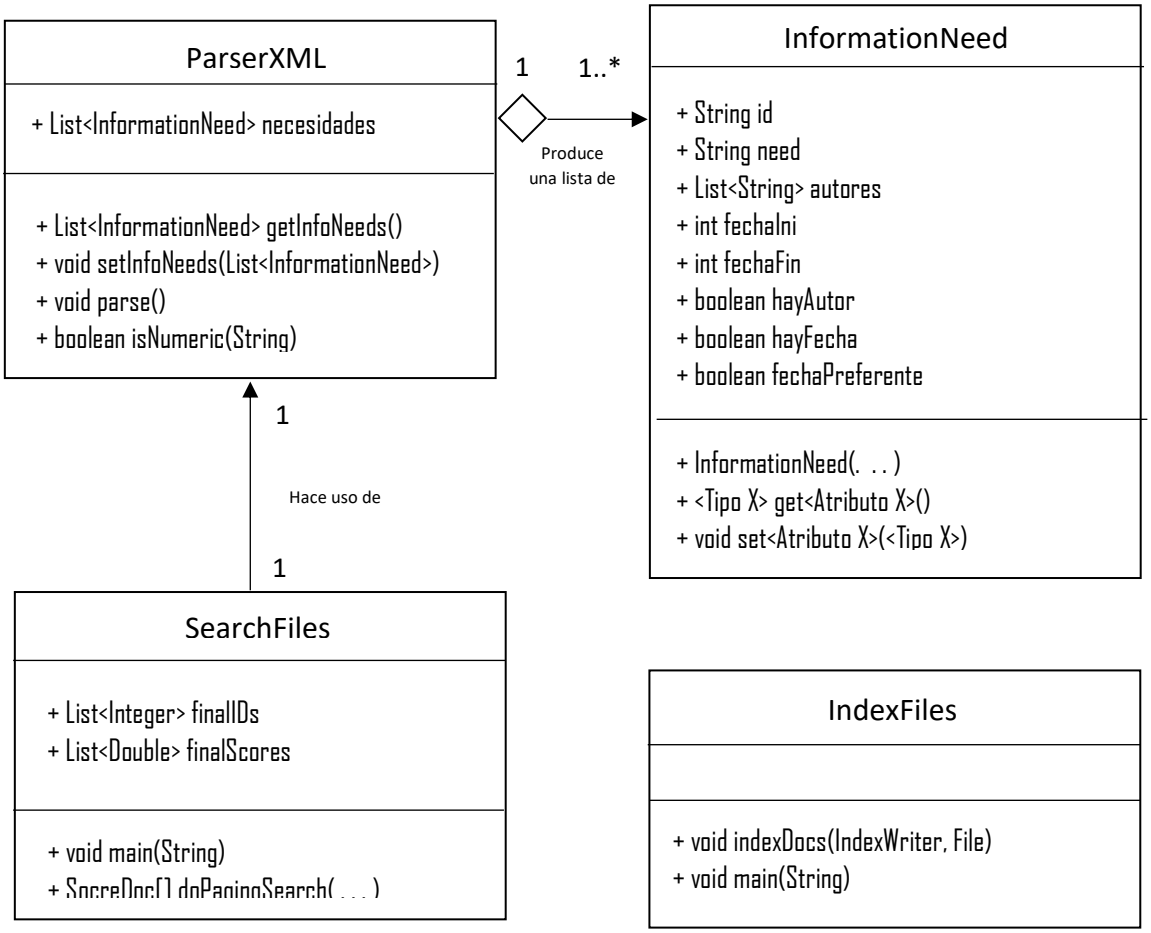
PABLO LUESIA - 698387

1.- Introducción

Este trabajo práctico de la asignatura de Recuperación de Información consiste en el diseño e implementación de un sistema de recuperación de información para posibles necesidades sobre la colección de documentos de Zaguán, repositorio de los trabajos finales de alumnos de la Universidad de Zaragoza.

En este primer informe se comentarán las decisiones de diseño primeras sobre este sistema.

2.- Diagrama de clases



El diagrama consta de cuatro clases básicas.

1) ***IndexFiles***: Es la case encargada de crear los índices según el proceso explicado posteriormente. Es independiente a las demás.

2) ***SearchFiles***: Es la clase encargada de llevar a cabo las búsquedas y de evaluar los resultados obtenidos en las mismas. Su funcionamiento también se relata posteriormente, y es dependiente con las clases que se comentan a continuación.

3) ***ParserXML***: Se trata de la clase que se emplea para transformar el fichero XML de necesidades en instancias manejables por Java.

4) ***InformationNeed***: Es la clase que instancia cada necesidad recuperada del fichero. Esta clase contiene, por un lado, el identificador y necesidad en sí (datos de los campos XML del fichero), así como atributos relacionados con las restricciones de la necesidad:

i.- Si hay restricción de fecha, se guarda una fecha inicial y final.

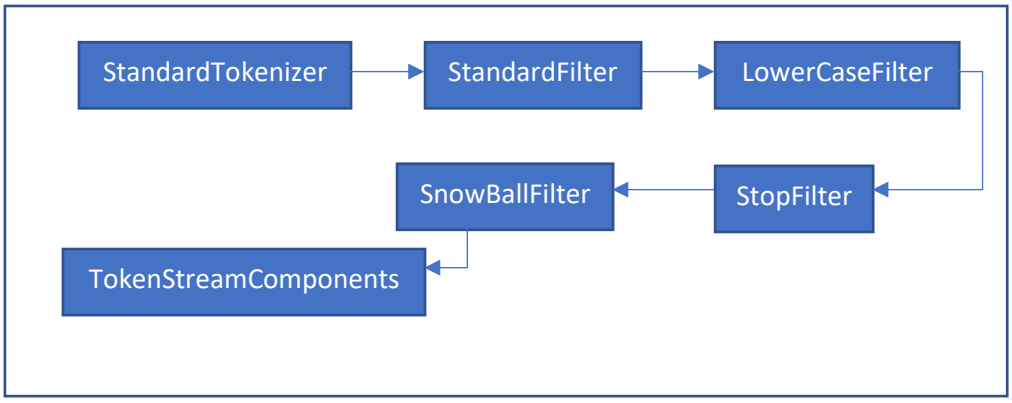
ii.- Si hay restricción de autoría, se guarda una lista de autores.

iii.- Si la fecha es preferente.

3.- Proceso de indexación. (IndexFiles)

A continuación, se indican la secuencia de acciones que realiza el proceso de indexación, así como cualquier detalle escogido que se considere deba ser mencionado en este trabajo.[1]

1) **Creación del directorio de los índices**. La primera decisión ha radicado sobre dónde crear dicho directorio, ante la posibilidad de hacerlo en disco o en RAM. Se ha escogido la opción de File System, disco, debido a su gran potencial en escalabilidad, evitando así cualquier posible límite de memoria.
2) **Elección del analizador**. En primera instancia, se parte del analizador básico de español que proporciona Lucene (SanishAnalyzer). Se razona esta decisión porque se cree difícil mejorar un analizador que se encuentra en una de las bibliotecas más importantes del ámbito. Por este motivo, se decide hacer uso de este analizador (en lugar de modificarlo o crear uno propio según algunas pautas del Anexo de la Práctica 1) y estudiar su funcionamiento para saber cómo llega a los resultados que obtiene.
    a. **Descripción**: Resumiendo brevemente, el analizador de Lucene cuenta con tres elementos clave en su funcionamiento:
        i. **Stopwords**: Se trata de una lista que, tanto en indexación como en búsqueda, son omitidas o ignoradas. En este listado, se podrían incluir preposiciones, conjunciones, artículos, pronombres, determinantes y otros elementos que no tienen significado/valor por sí mismos.
        ii. **Stemming**: Se trata del método que permite obtener la raíz semántica de una palabra. Se emplea para que una palabra pueda ser relacionada con cualquier otra de su familia, ampliando así el ratio de búsqueda. [2]
        iii. **Modelo de espacio vectorial**: El analizador emplea este modelo algebraico para filtrar, indexar y calcular relevancia de la diferente información que se va obteniendo, es un espacio multidimensional (haciendo operaciones sobre ángulos en dicho espacio).
    b. **Funcionamiento:** En la imagen siguiente, se muestra el flujo de funcionamiento del SpanishAnalyzer de Lucene. A continuación, se explica paso a paso:



[1] Se ha decidido explicar paso a paso el proceso en lugar de emplear diagramas de secuencia para facilitar así la adición de notas, comentarios o explicaciones respecto a las decisiones tomadas.
[2] Por ejemplo, tras aplicar la eliminación de stopwords, y stemming a "`Estamos interesados en` recopilar todo tipo de información acerca de las características de la frontera entre Francia y España." se obtendría "interesad recopilar tipo informacion acerc caracteristic fronter franci españ ".

i. **StandardTokenizer:** Transforma la entrada del Reader (flujo de caracteres) en tokens. Para ello, transforma cada palabra en un token, separada según el estándar de Unicode Text Segmentation.
ii. **StandardFilter:** Normaliza los tokens extraídos del StandarTokenizer.
iii. **LowerCaseFilter:** Normaliza los tokens a minúsculas.
iv. **StopFilter:** Se carga la lista de stopwords. En este caso, es una lista de palabras en español. Estas palabras son comunes y no aportan datos relevantes a la búsqueda (preposiciones, artículos…).
v. **SnowBallFilter con SpanishStemer:** Se realiza el stemming. Para ello se utiliza SnowBallFilter, que permite realizar el stemming con cualquier stemmer generado por Snowball. En este caso es el stemmer español.
vi. **TokenStreamComponents:** Se encapsulan todos los componentes de salida del flujo de tokens.

3) **Creación de los índices**. Tras haber hecho los ajustes necesarios y haber creado los elementos indispensables en el proceso, se pasa a la creación de los índices. Para ello, se hace uso de las mismas estructuras y técnicas de las que se dispuso en las prácticas de la asignatura, creando documentos (clases ***Document***, ***DocumentBuilder***…) y haciendo uso de las estructuras de *nodos* y *listas de nodos* para transformar el XML recibido en datos manejables por Java.

4) **Elección de los campos a indexar**. Para elegir que campos son indexados y guardados en los documentos, se ha estudiado la colección de documentos que se proporcionaba, y tomado la siguiente decisión.
   a. Se guardará la ruta del fichero (para tener siempre la referencia origen de los resultados), aunque esta no será analizada, por lo que se guardará como un campo de tipo ***StringField***.
   b. Se guardarán tres campos relativos al trabajo, que serán analizados, y a los cuales se les asignará un peso para su posterior valoración. (Ver Proceso de búsqueda). Como han de ser analizados, serán campos ***TextField***.
      i. <dc:title> : Contiene el título del trabajo.
      ii. <dc:subject> : Contiene las palabras clave del trabajo.
      iii. <dc:description> : Contiene la descripción extensa del trabajo.
   c. Se guardará un campo numérico relativo a la fecha de origen, que se empleará posteriormente en el análisis de rangos temporales. Como tendrá un rango numérico, se empleará el tipo ***DoublePoint***.
      i. <dc:date> : Contiene la fecha del trabajo.
   d. Se guardará un campo de texto que será analizado, con los diferentes autores de cada trabajo, para filtrar las búsquedas. Como ha de ser analizado, será campo ***TextField***.
      i. <dc:creator> : Contiene el autor del trabajo.

   En definitiva, se crearán 5 índices: Título, Temas, Descripción, Fecha y Creador. Los tres primeros para buscar la temática, y los otros dos para las posibles restricciones del sistema.

5) **Escritura de los índices**. Finalmente, se guardarán estos índices en el directorio indicado por el usuario.

```
/* Campo de fecha */
if(nodo.getElementsByTagName("dc:date") != null && nodo.getElementsByTagName("dc:date").getLength()>0){
    for(int i = 0; i<nodo.getElementsByTagName("dc:date").getLength(); ++i){
        date = new DoublePoint("date", Double.valueOf(nodo.getElementsByTagName("dc:date").item(i).getTextContent()));
        doc.add(date);
    }
}

/* Campo de autor */
if(nodo.getElementsByTagName("dc:creator") != null && nodo.getElementsByTagName("dc:creator").getLength()>0){
    for(int i = 0; i<nodo.getElementsByTagName("dc:creator").getLength(); ++i){
        autors = new TextField("creator", nodo.getElementsByTagName("dc:creator").item(i).getTextContent(), Field.Store.YES);
        doc.add(autors);
    }
}
```

Ejemplo de creación de los campos a indexar.

4.- Proceso de búsqueda y valoración (ranking)

Tras el proceso de indexación, y la obtención de los índices mediante Lucene, se da paso al proceso de búsqueda, para el cual se leen las necesidades de información obtenidas en clase y se ejecuta una búsqueda con valoración sobre los resultados obtenidos. Por los mismos motivos que previamente, se explica a continuación una secuencia de pasos relevantes en el proceso.

1) **Lectura del fichero de necesidades**: El primer paso parte de la lectura del fichero de necesidades y su transformación en una lista de objetos manejables por Java, a través de una clase que tratase el XML creada personalmente. Esta transformación facilita enormemente la iteración, ordenación, extracción de información, etc. [3]
    a. Cabe destacar que estas necesidades pueden incluir restricciones de fecha y autoría, que son tratadas de forma especial. [**Ver ANEXO para restricciones**]

2) **Obtención de resultados para cada necesidad**: Tras la obtención de esta lista, se itera sobre ella para obtener los resultados esperados del sistema. Los siguientes pasos se realizarán individualmente para cada necesidad.

3) **[ITER-1] Generación de "*queries*" para la consulta**: Siguiendo la filosofía citada previamente, se llevarán a cabo tres consultas sobre cada necesidad, una por cada uno de los tres campos indexados previamente (a saber, <dc:title>, <dc:subject> y <dc:description>), de los cuales se obtendrán tres tablas de resultados diferentes, según cada búsqueda en esos campos. El número de resultados NO está limitado.
    a. Cabe destacar que, a estas tres consultas, se les puede haber añadido algún tipo de restricción temporal o de autores. [**Ver ANEXO para restricciones**]

4) **[ITER-2] Cálculo de las puntuaciones obtenidas**: Teniendo en cuenta los resultados de las 3 consultas, se pasará a generar una puntación global de los resultados, siguiendo un algoritmo que responde de esta manera:

$$score_{final}(d) = score_{titulo}(d) + 0.5 \times score_{temas}(d) + 0.25 \times score_{descripcion}(d)$$

    Esta puntuación se debe a una decisión de equipo entorno a la siguiente filosofía:

    - El usuario leerá primero los títulos, por lo que es lo más relevante.
    - Si el título parece conformarle, mirará los tópicos o temas del trabajo, dándoles una relevancia media.
    - Si estos parecen interesarle, pasará a leer la descripción. Por ello, la relevancia es la menor.
    -

5) **[ITER-3] Ordenación de los documentos**: Tras obtener una puntuación global de cada documento, se ordena el resultado, para así poder mostrarlo, guardarlo y devolver aquella información que sea necesaria.

[3] Se ha empleado una implementación propia del tratador de XML debido a que los miembros del grupo, implementadores del sistema, ya contaban con una clase similar, y ha agilizado y facilitado el proceso, así como afianzado el resultado.

5.- Resultados obtenidos

Tras esto, se prueba el funcionamiento del sistema, que parece responder adecuadamente. Tomando como ejemplo la primera búsqueda, cabe destacar que los resultados que obtiene contienen en el título términos como “visión”, “computador”, “robótica”, o similares. En general, esto ocurre con todas las necesidades. Se adapta, además, a las restricciones temporales y de autoría que se solicitan.

La primera conclusión que se observa es que el método de ranking elegido se plasma en los resultados, y se ve su precisión: Valora muy positivamente que el fichero contenga un título apropiado y conciso, y da valor a que los tópicos elegidos sean precisos.

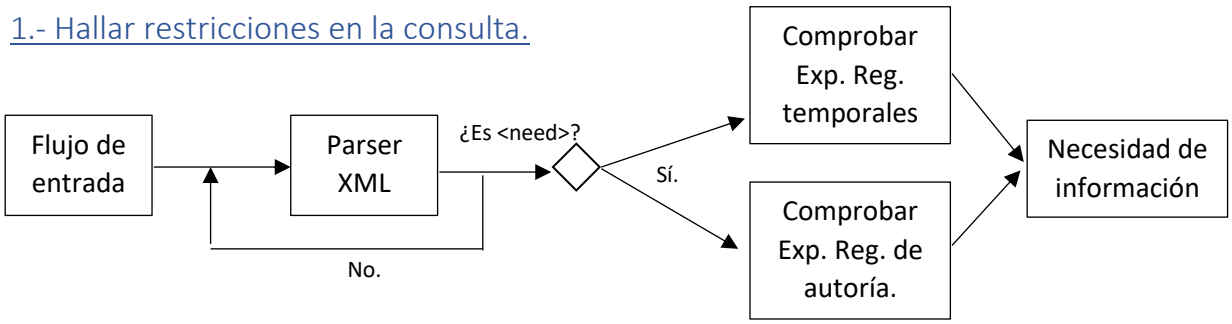
Es posible que haya documentos con una descripción bastante buena, pero es arriesgado darle el mismo peso que a otros campos, puesto que, en una búsqueda real en un repositorio, un usuario no va a emplear tiempo en leer algo cuyo resumen, título o enlace no ha parecido cubrir sus necesidades.

ANEXO I – Restricciones temporales y de autoría.

Una de las principales prestaciones que debe proporcionar el sistema de recuperación es el posible filtrado de consultas según un rango de valores para la fecha, o según una petición de autoría. Para ello, ha habido que intervenir en la indexación y búsqueda. Sobre el primero de los dos, se ha explicado previamente cómo se han indexado los campos referentes a estas restricciones.

Respecto a la búsqueda, ha habido dos pasos cruciales: la obtención de restricciones dentro de la necesidad, y la adaptación de la *query* posterior.

1.- Hallar restricciones en la consulta.



La estructura básica de esta funcionalidad parte del uso de expresiones regulares para determinar si existen limitaciones temporales o de autoría, ya que no existe una forma óptima de saber si un nombre propio es una persona; o un número, una fecha. Es por ello, que se parte de una serie de patrones básicos, que posteriormente podrían irse mejorando y complementando.

1.1.- Expresiones regulares para fechas:

- ***.*a partir .* [0-9]+.**** (a partir de 2014, a partir del año 2014, a partir del año académico 2014…)
- ***.*hasta [0-9]+.**** (hasta 2014)
- ***.*desde [0-9]+.**** (desde 2014)
- ***.*entre [0-9]+ y [0-9]+.**** (entre 2014 y 2016)
- ***.*entre el [0-9]+ y el [0-9]+.**** (entre el 2014 y el 2016)
- ***.*antes de [0-9]+.**** (antes de 2013)
- ***.*antes del [0-9]+.**** (antes del 2014)
- ***.*del.*[0-9]+.**** (del 2014, del año 2015, del año académico pasado 2011…)

Cabe destacar que el repertorio de expresiones regulares que puede contemplar el castellano es casi tan interminable como el idioma en sí. Se han considerado estas por ser las más usuales y simples de tratar, pero se sobreentiende que se podrían añadir muchas más que cubriesen otro tipo de expresiones de significados similares, pero con distinta sintaxis. No obstante, con las actuales se ha logrado el objetivo de comprender la dificultad y preparar el sistema frente a estas imposiciones.

En cuanto a la técnica, si una de estas expresiones aparecía, según si fuese de anterioridad, posterioridad, igualdad o rango, se le asignaba a la necesidad de información una fecha de origen y otra de destino. Si sólo se dispone de una de las dos, la otra se rellena por defecto. La fecha inicial por defecto sería 1900 y la final, 2020.

1.2.- Expresiones regulares para autoría:

- ***.escrito por.*** (escrito por Pedro)
- ***.hecho por.*** (hecho por Daniel)
- ***.realizado por.*** (realizado por Pablo)
- ***.escribi.*** (lo escribió Irene, lo escribio Laura)
- ***.hizo.*** (lo hizo Carlos)
- ***.realiz.*** (lo realizó Adriana, lo realizo Rafael)

Siguiendo la misma filosofía que con las fechas, se ha tenido que reducir el número de expresiones que llega a tratar el sistema, debido a que es imposible poder contemplar todos los casos, situaciones y posibilidades del lenguaje. Sin embargo, de nuevo, con estas expresiones se puede llegar a tratar la base del problema y comprobar las limitaciones que impone.

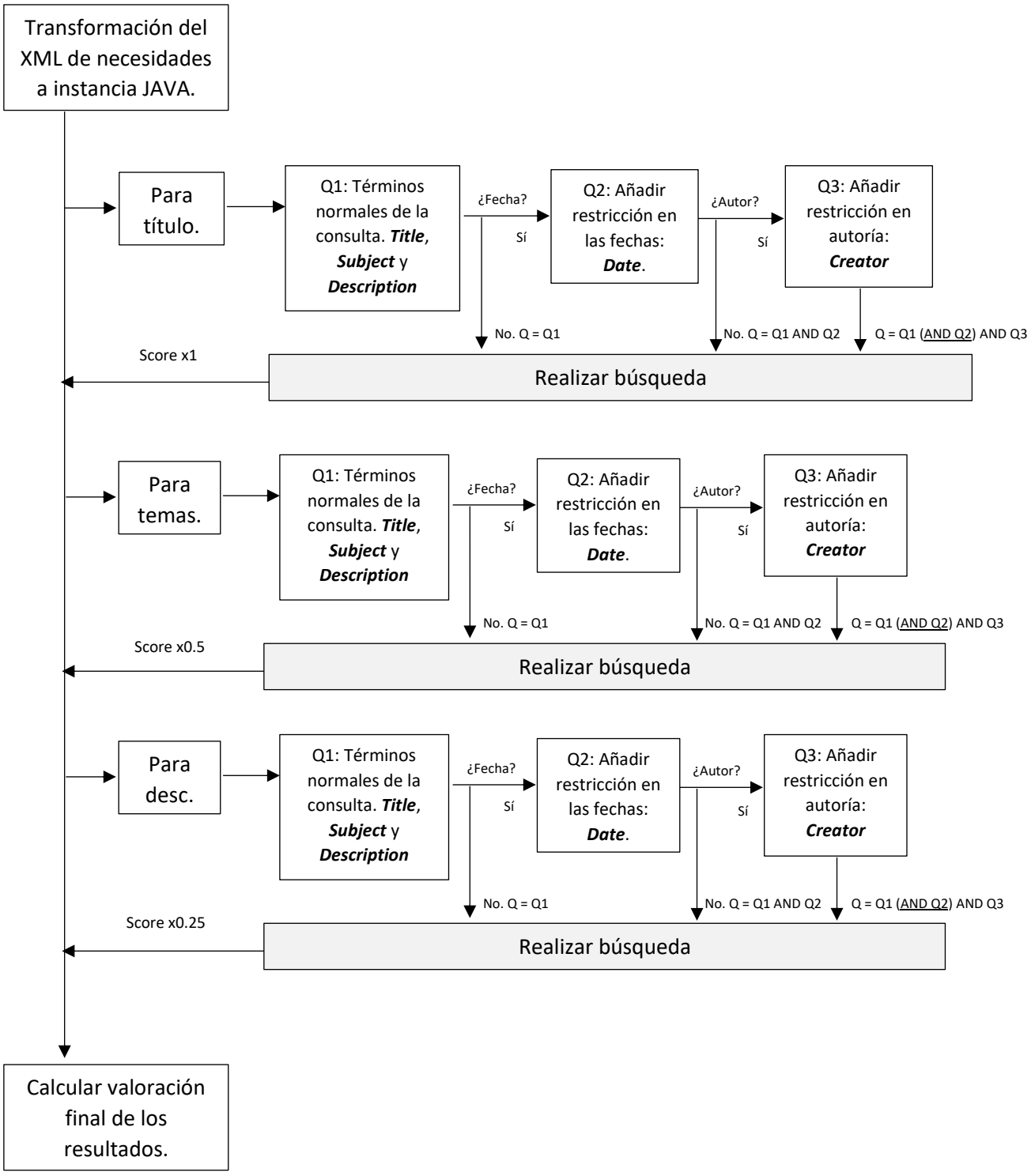
1.3.- Preferencia u obligatoriedad:

Hay un concepto que se considera importante en el análisis de las franjas temporales impuestas y en la autoría, y radica sobre si esos detalles deben ser impuestas o son sólo preferentes. Para ello, se ha hecho uso de expresiones regulares igualmente. Si la necesidad contiene alguna de las siguientes palabras o raíces, se entenderá que la fecha es una adición recomendable PERO NO OBLIGATORIA: "***prefer***", "***a poder ser***", "***si es posible***", "***si puede ser***". De nuevo, se podría ampliar este repertorio, aunque estas son consideradas las más importantes.

Entrando en el código, esto se traduce a forzar la *query* a ser *BooleanClause.Occur.**MUST*** o *BooleanClause.Occur.**SHOULD***. Lógicamente, la preferencia proporciona más resultados, mientras que la obligatoriedad reduce en gran medida los documentos devueltos.

En contraposición, **la autoría se toma siempre como obligatoria**.

Básicamente, esto transforma la búsqueda en una serie de decisiones, que acaban por seguir el siguiente flujo:



1.4.- Pruebas de este sistema.

Tomando como ejemplo la primera necesidad, que cita como sigue, se harán ciertas modificaciones.

"Estoy interesado en tesis doctorales sobre robotica y la vision computador a partir del 2010".

Al lanzar el buscador, los resultados obtenidos son:

- 2037 resultados.
- Tres primeros ficheros:
    - 5621 (score = 27.257)
    - 11855 (score = 23.357)
    - 10525 (score = 20.373)

"Estoy interesado en tesis doctorales sobre robotica y la vision computador a partir del 2014".

Al lanzar el buscador, los resultados obtenidos son:

- 1422 resultados.
- Tres primeros ficheros:
    - 31424 (score = 16.228)
    - 58574 (score = 15.543)
    - 13528 (score = 14.176)

"Estoy interesado en tesis doctorales sobre robotica y la vision computador a partir del 2014 y escritos por Daniel".

Al lanzar el buscador, los resultados obtenidos son:

- 33 resultados.
- Tres primeros ficheros:
    - 56483 (score = 16.247)
    - 58339 (score = 13.986)
    - 61126 (score = 13.622)

Abriendo esos tres últimos ficheros, se comprueba.

`<dc:date>2016</dc:date>` y `<dc:creator>Checa Nebot, Daniel</dc:creator>`

`<dc:date>2016</dc:date>` y `<dc:creator>Daniel Vicente Moya</dc:creator>`

`<dc:date>2016</dc:date>` y `<dc:creator>Filippini, Daniel</dc:creator>`

En efecto, ha tenido en cuenta los filtros y restricciones, y ha funcionado correctamente.