Euskal Herriko Unibertsitatea / Universidad del País Vasco

Departamento de Lenguajes y Sistemas Informáticos

Interfaz para facilitar la edición y visualización de kybots

Proyecto presentado por

Daniel Candelas Peral

Director: German Rigau Claramunt
Codirector: Aitor Soroa Echave

Julio de 2013, en Donostia

Resumen

Este documento describe la interfaz web creada para facilitar la interacción con el *Mining Module* (Módulo de minería) del sistema de extracción de información KYOTO. Hasta ahora, la interacción con dicho módulo requería cierto nivel técnico y además era necesario tenerlo instalado en un entorno local. Ahora, en cambio, se alberga en un servidor web, por lo que se accederá a la interfaz mediante cualquier navegador. El núcleo de esta aplicación sigue siendo el *Mining Module*, pero ahora el usuario puede interactuar con el sistema de una forma más sencilla e intuitiva. De esta forma se podrán editar y visualizar los kybots y los resultados de una forma mucho más rápida y eficiente.

Palabras clave:

1. Minería de texto
2. Kybots
3. Interfaz Web
4. KYOTO
5. Mining Module

Índice general

Índice de figuras

I. CAPÍTULO

Introducción

En este documento se presenta el proyecto de fin de carrera realizado por el alumno Daniel Candelas Peral para obtener el título de ingeniero en informática. Se ha llevado a cabo a petición del departamento de lenguajes y sistemas informáticos de la facultad de informática de la UPV/EHU, y ha sido dirigido por German Rigau Claramunt y codirigido por Aitor Soroa Echave.

En este proyecto se presenta una interfaz para agilizar y enriquecer la interacción con el *Mining Module* (Módulo de minería) del proyecto KYOTO (Piek *et al.* (2013))[1]. Este proyecto se basa en la extracción de información de documentos de un campo específico (medicina, medio ambiente, etc.) para su posterior consulta. La razón principal por la que se ha creado esta interfaz es la de facilitar a los usuarios (lingüistas y expertos del campo que se esté tratando) la interacción con el módulo de minería. A continuación se muestran varios ejemplos:

- La instalación y configuración del módulo de minería debe hacerse en Linux. Para ello, se tienen que ir instalando los distintos componentes (entorno, BBDD, etc.) manualmente, los cuales pueden causar problemas de compatibilidad o diversos errores si no se instala la versión adecuada. Con la interfaz, en cambio, el usuario no tendrá que hacer ningún tipo de instalación. Ésta, junto con el módulo, se encuentra alojados en un servidor. Para acceder a ella, el usuario sólo necesitara un

[1] http://www.kyoto-project.eu/

navegador web y conexión a internet.

- Actualmente, todas las operaciones se realizan mediante comandos en el terminal, tarea poco intuitiva y muy propensa a dar fallos si no se teclea correctamente toda la sentencia. Aquí vemos dos ejemplos muy simples:

  ./kybot_load.pl –container-name kybots_en –force /kyoto/kybots_en
  ./kybot_run.pl –container-name docs_en –kybot-container-name kybots_en kprofile1

  Con la interfaz, en cambio, todas las acciones se realizan mediante el navegador y de una forma intuitiva y visual, eliminando así la interacción con complejos comandos. Además, se han añadido muchas más funcionalidades para agilizar todo el proceso de creación y modificación de reglas (kybots) del módulo de minería.

- Los documentos introducidos al sistema son procesados y convertidos a archivos KAF (Kyoto Annotation Format) (Bosma *et al.* (2009)), que son con los que interacciona el usuario. La figura I.1 muestra una pequeña sección de un documento KAF. Como puede apreciarse, son unos archivos en formato XML, que, tras haber pasado por diferentes procesadores lingüísticos, representan texto enriquecido. Estos pueden resultar poco legibles y muy extensos a medida que van creciendo. La interfaz, en cambio, los muestra en un formato mucho más compacto, mostrando únicamente la información necesaria y en un formato fácil de entender.

- Los kybots, otro compenente básico del sistema que sirve para extraer hechos de los documentos KAF, es otro de los componentes que más a menudo manipulará el usuario. En la figura I.2 puede apreciarse cómo es un kybot bastante simple. A medida que sean más complejos, estos irán creciendo y su estructura se irá haciendo más compleja, resultando así difícil de editar y visualizar para muchos usuarios. Al igual que con los documentos KAF, la interfaz mostrará los kybots en otro formato mucho más amigable, y el usuario no tendrá que modificar directamente el fichero físico, evitando así la posibilidad de cometer errores sintácticos a la hora de construir los XML. Por lo tanto, los resultados que producen los kybots, similares a los documentos KAF, se visualizarán al momento y en un formato mucho más fácil de entender.

```
<KAF xml:lang="es">
  <kafHeader>
    <linguisticProcessors layer="terms">
      <lp name="Freeling" version="3.0" timestamp="2012-12-09T10:22:17Z"></lp>
    </linguisticProcessors>
  <linguisticProcessors layer="terms"><lp name="ukb" version="0.1.6" timestamp="2012-12-09T10:22:47Z"></lp></linguisticProcessors>
<linguisticProcessors layer="onto">
          <lp name="OntoTagger" timestamp="Sun-Dec--9-10:22:51-2012" version="v3"/>
       </linguisticProcessors>
    </kafHeader>
  <text>
    <wf wid="w1" sent="1" offset="0" lenght="8">aparecer</wf>
    <wf wid="w2" sent="1" offset="8" lenght="1">#</wf>
    <wf wid="w3" sent="1" offset="9" lenght="2">El</wf>
    <wf wid="w4" sent="1" offset="12" lenght="6">nombre</wf>
  </text>
  <terms>
    <term tid="t1_1_es_" lemma="aparecer" pos="VMN0000">
      <span>
        <target id="w1"></target>
      </span>
    <externalReferences><externalRef resource="spwn3.0" reference="eng-30-02744061-v" confidence="0.176132">
                    <externalRef confidence="1.0" reference="eng-30-02604760-v"  reftype="baseConcept" resource="wn30g"/>
                    <externalRef confidence="1.0" reference="Kyoto#be-eng-3.0-02604760-v" reftype="sc_subClassOf" resource="ontology">
                        <externalRef reftype="SubClassOf" reference="Kyoto#be-eng-3.0-02604760-v" status="implied"/>
                        <externalRef reftype="SubClassOf" reference="Kyoto#verb_stative"/>
                        <externalRef reftype="SubClassOf" reference="DOLCE-Lite.owl#perdurant" status="implied"/>
                        <externalRef reftype="SubClassOf" reference="DOLCE-Lite.owl#particular" status="implied"/>
                    </externalRef>
                </externalRef><externalRef resource="spwn3.0" reference="eng-30-00528990-v" confidence="0.124892">
                </externalRef></externalReferences></term>
    <term tid="t2_1_es_" lemma="el" pos="NP00000">
      <span>
        <target id="w3"></target>
      </span>
    </term>
    <term tid="t3_1_es_" lemma="nombre" pos="NCMS000">
      <span>
        <target id="w4"></target>
      </span>
    <externalReferences><externalRef resource="spwn3.0" reference="eng-30-06333653-n" confidence="0.215026">
                    <externalRef confidence="1.0" reference="eng-30-00031921-n"  reftype="baseConcept" resource="wn30g"/>
                    <externalRef confidence="1.0" reference="Kyoto#quality-eng-3.0-04723816-n" reftype="sc_hasQuality" resource="ontology">
                        <externalRef reftype="SubClassOf" reference="Kyoto#quality-eng-3.0-04723816-n" status="implied"/>
                        <externalRef reftype="SubClassOf" reference="DOLCE-Lite.owl#quality"/>
```

Figura I.1: Fragmento de un archivo KAF

```
<Kybot id="magnitud">

<variables>
   <var name="A" type="term" pos ="N"/>
   <var name="B" type="term" lemma="crecer | decrecer"/>
   <var name="C" type="term" pos="D"/>
   <var name="D" type="term" lemma="*%"/>
 </variables>

 <relations>
   <root span="A"/>
   <rel span="B" pivot="A" direction="following" immediate="true" />
   <rel span="C" pivot="B" direction="following" />
   <rel span="D" pivot="C" direction="following" />
 </relations>

 <events>
   <event target="$A/@tid" lemma="$A/@lemma" pos="$A/@pos"/>
   <role  target="$B/@tid" rtype="accion" lemma="$B/@lemma"/>
   <role  target="$D/@tid" rtype="cantidad" lemma="$D/@lemma"/>
 </events>

</Kybot>
```

Figura I.2: Estructura de un kybot simple

Por tanto, el objetivo será crear una interfaz para que los usuarios, partiendo de kybots simples, vayan editándolos y enriqueciéndolos de una forma sencilla y en la que vayan viendo los resultados que producen con cada cambio que realizan. De este modo, será posible la creación de kybots complejos de una forma incremental e interactiva en mucho menos tiempo que hasta ahora.

II. CAPÍTULO

Antecedentes

La interfaz va a ser desarrollada para uno de lo módulos del sistema KYOTO, por lo que es necesario saber qué es y cómo funciona para entender los beneficios que reporta nuestra interfaz. Así pues, en esta sección se explicarán las caraterísticas más importantes del proyecto KYOTO, y se analizará en más profundidad el módulo y los componentes que vayan a interaccionar con la interfaz.

II.1. Proyecto KYOTO

KYOTO[1] (*Knowledge Yielding Ontologies for Transition-Based Organization*) (Piek *et al.* (2013)) es un sistema de minería de texto, y su objetivo principal es crear una plataforma en la que se comparta información y conocimiento de un campo en concreto (medicina, medio ambiente, etc.). Esto se logra analizando y relacionando todos los documentos introducidos al sistema, de tal forma que los usuarios tendrán acceso a las relaciones semánticas, hechos, patrones lingüísticos y otro tipo de información extraída en esos procesos. La figura II.1 muestra la arquitectura general del sistema y también el flujo de las actividades más importantes. A continuación se explicarán brevemente los 4 pasos más importantes de todo el proceso:

1. El primer paso será introducir los documentos (PDF, HTML, DOC,

[1] http://www.kyoto-project.eu/

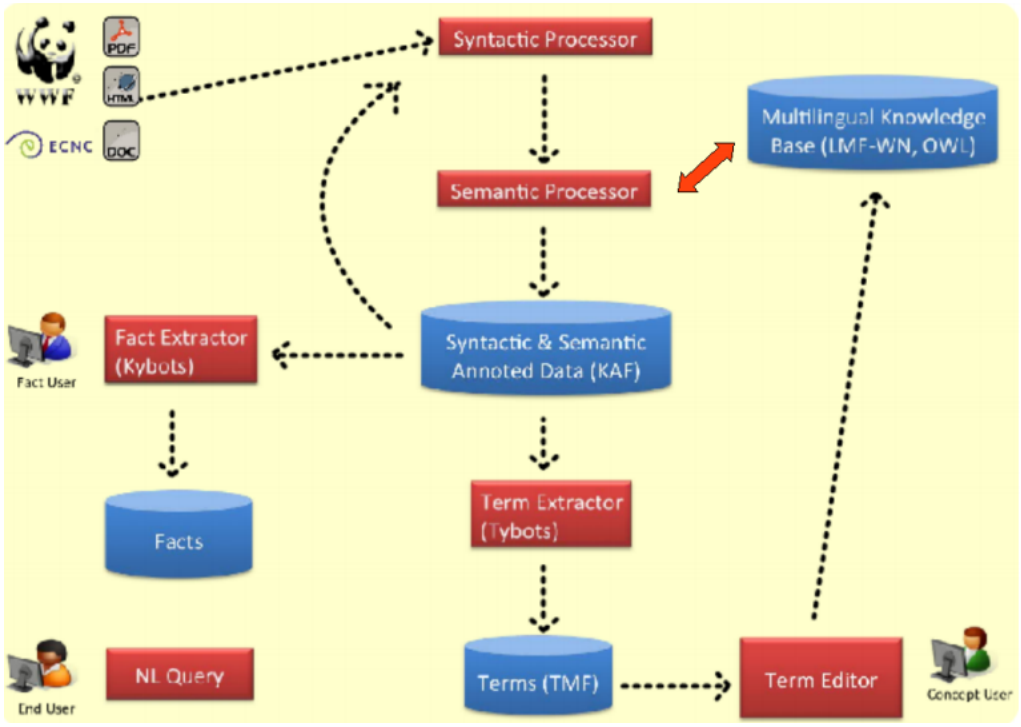


Figura II.1: Arquitectura general de KYOTO

TXT) del dominio deseado en el sistema. Al hacer esto, serán procesados por varios procesadores lingüísticos y generarán unos documentos anotados llamados KAF, que son una versión enriquecida de los textos introducidos al comienzo. Todos los módulos de KYOTO usarán estos archivos KAF en vez de las fuentes originales introducidas. Un poco más adelante se explicará en más profundidad el formato KAF.

2. Tras este primer paso, se extrae el conocimiento conceptual de los KAFs. El conocimiento conceptual se refiere a las propiedades y relaciones entre los términos del dominio. Por ejemplo, *ave migratoria* se refiere a determinado tipo del concepto *ave*. Los tybots analizarán los documentos KAF para obtener un modelo de los términos y conceptos más importantes de los textos.

3. Después, se obtiene el conocimiento factual, o de hechos. Éste es expresado mediante propiedades y relaciones entre diferentes entidades que interactúan en los textos del dominio. Por ejemplo, *la población de aves migratorias del Amazonas ha disminuido a la mitad en los últimos 4 años*, describe que las aves que usan el Amazonas para alimentarse ha disminuido en cierta magnitud en un particular periodo de tiempo. De este tipo de extracción se encargan unos scripts en XML llamados kybots. Estos, al igual que los tybots, cogerán los documentos KAF y a continuación analizarán las relaciones y los hechos más relevantes para almacenarlos en el sistema.

   Estos tres pasos, por supuesto, podrán realizarse tantas veces como haga falta para ir añadiendo más documentos y extrayendo más hechos, y por lo tanto, más conocimiento al sistema.

4. El último paso, que a fin de cuentas es la meta de todo el proyecto KYOTO, es que los usuarios accedan a toda esta información y conocimiento recopilado durante todos los procesos anteriores.

Después de haber explicado la estructura y funcionamiento general del sistema KYOTO, se analizarán los documentos KAF y los perfiles kybot: dos elementos del proyecto KYOTO que interaccionarán con nuestra interfaz.

```
<KAF xml:lang="es">
  <kafHeader>
    <linguisticProcessors layer="terms">
      <lp name="Freeling" version="3.0" timestamp="2012-12-09T10:22:17Z"></lp>
    </linguisticProcessors>
  <linguisticProcessors layer="terms"><lp name="ukb" version="0.1.6" timestamp="2012-12-09T10:22:47Z"></lp></linguisticProcessors>
<linguisticProcessors layer="onto">
           <lp name="OntoTagger" timestamp="Sun-Dec--9-10:22:51-2012" version="v3"/>
       </linguisticProcessors>
  </kafHeader>
  <text>
    <wf wid="w1" sent="1" offset="0" lenght="8">aparecer</wf>
    <wf wid="w2" sent="1" offset="8" lenght="1">#</wf>
    <wf wid="w3" sent="1" offset="9" lenght="2">El</wf>
    <wf wid="w4" sent="1" offset="12" lenght="6">nombre</wf>
  </text>
  <terms>
    <term tid="t1_1_es_" lemma="aparecer" pos="VMN0000">
      <span>
        <target id="w1"></target>
      </span>
    <externalReferences><externalRef resource="spwn3.0" reference="eng-30-02744061-v" confidence="0.176132">
                  <externalRef confidence="1.0" reference="eng-30-02604760-v"  reftype="baseConcept" resource="wn30g"/>
                  <externalRef confidence="1.0" reference="Kyoto#be-eng-3.0-02604760-v" reftype="sc_subClassOf" resource="ontology">
                      <externalRef reftype="SubClassOf" reference="Kyoto#be-eng-3.0-02604760-v" status="implied"/>
                      <externalRef reftype="SubClassOf" reference="Kyoto#verb_stative"/>
                      <externalRef reftype="SubClassOf" reference="DOLCE-Lite.owl#perdurant" status="implied"/>
                      <externalRef reftype="SubClassOf" reference="DOLCE-Lite.owl#particular" status="implied"/>
                  </externalRef>
              </externalRef><externalRef resource="spwn3.0" reference="eng-30-00528990-v" confidence="0.124892">
              </externalRef></externalReferences></term>
    <term tid="t2_1_es_" lemma="el" pos="NP00000">
      <span>
        <target id="w3"></target>
      </span>
    </term>
    <term tid="t3_1_es_" lemma="nombre" pos="NCMS000">
      <span>
        <target id="w4"></target>
      </span>
    <externalReferences><externalRef resource="spwn3.0" reference="eng-30-06333653-n" confidence="0.215026">
                  <externalRef confidence="1.0" reference="eng-30-00031921-n"  reftype="baseConcept" resource="wn30g"/>
                  <externalRef confidence="1.0" reference="Kyoto#quality-eng-3.0-04723816-n" reftype="sc_hasQuality" resource="ontology">
                      <externalRef reftype="SubClassOf" reference="Kyoto#quality-eng-3.0-04723816-n" status="implied"/>
                      <externalRef reftype="SubClassOf" reference="DOLCE-Lite.owl#quality"/>
```

Figura II.2: Fragmento de un archivo KAF

II.1.1. Documentos KAF (*Kyoto Annotation Format*)

Los módulos de KYOTO no usan directamente los documentos introducidas al sistema. Primero, esos documentos deben pasar por unos procesadores lingüísticos, que crean los documentos KAF: documentos XML en los que se representa el texto enriquecido con información lingüística. Toda esta información estará estructurada en diferentes capas a lo largo del documento, y en cualquier momento podrán añadirse nuevas capas si es necesario, por lo que los KAF son muy flexibles y expandibles.

Cualquier documento KAF contendrá las siguientes capas de información lingüística:

- *Word forms* (formas de palabra) *tokenizadas* y segmentadas extraídas directamente del texto.
- *Terms* (términos) lematizados que corresponden a uno o varios *word forms*.

- *Chunks* (sintagmas), que abarcan un conjunto de términos.
- Dependencias sintácticas entre los sintagmas.
- Roles semánticos para los sintagmas.
- Hechos o eventos obtenidos de los kybots.

A pesar de reportar muchos beneficios al sistema, el formato KAF también tiene algunos inconvenientes: Pueden llegar a ser bastante grandes y complejos, y además el hecho de estar en un formato que no resulta fácil de interpretar dificulta la compresión a los usuarios. La figura II.2 muestra un pequeño fragmento de uno de estos documentos.

II.1.2. Perfiles Kybot

Los kybots son scripts en formato XML cuyo objetivo es extraer información de los documentos KAF. Para ello, los kybots analizan documentos KAF en busca de determinados patrones. La figura II.3 muestra cómo es un kybot simple. Por ejemplo, ese kybot ha sido definido para que busque patrones como *X creció un Y %* o *X decreció alrededor del Y %*, y generar un evento cada vez que se cumpla dicho patrón en los textos en los que ha buscado. En este caso, además, por cada evento generado, se han añadido dos roles al evento, los cuales sirven para mostrar más información a la hora de obtener el resultado. En este caso el primer rol indica qué acción se ha llevado a cabo en el evento (incremento, decremento), y el segundo, la cantidad, que sería el valor de Y.

Como puede apreciarse, el kybot está compuesto por tres módulos principales, los cuales analizaremos con más profundidad a continuación:

- Variables
- Relaciones entre las variables
- Formato del evento

II.1.2.1. Variables

En esta parte, se definen las entidades implicadas. En el ejemplo se han definido cuatro:

```
<Kybot id="magnitud">

<variables>
   <var name="A" type="term" pos ="N"/>
   <var name="B" type="term" lemma="crecer | decrecer"/>
   <var name="C" type="term" pos="D"/>
   <var name="D" type="term" lemma="*%"/>
 </variables>

 <relations>
   <root span="A"/>
   <rel span="B" pivot="A" direction="following" immediate="true" />
   <rel span="C" pivot="B" direction="following" />
   <rel span="D" pivot="C" direction="following" />
 </relations>

 <events>
   <event target="$A/@tid" lemma="$A/@lemma" pos="$A/@pos"/>
   <role  target="$B/@tid" rtype="accion" lemma="$B/@lemma"/>
   <role  target="$D/@tid" rtype="cantidad" lemma="$D/@lemma"/>
 </events>

</Kybot>
```

Figura II.3: Estructura de un kybot

```
<variables>
   <var name="A" type="term" pos ="N"/>
   <var name="B" type="term" lemma="crecer | decrecer"/>
   <var name="C" type="term" pos="D"/>
   <var name="D" type="term" lemma="*%"/>
 </variables>
```

Figura II.4: Definición de las variables

- A: La categoría gramatical del término debe ser sustantivo común.
- B: El lema puede ser *crecer* o *decrecer*.
- C: La categoría gramatical del término debe ser determinante.
- D: El lema debe terminar con el signo %.

Cada variable puede tener los siguientes atributos:

- *name*: Nombre de la variable.
- *type*: Tipo de la variable (término, dependencia, sintagma, etc)
- *lemma*: Lema del término. Pueden ponerse varios usando el separador |, que indica opcionalidad. El asterisco indica que cualquier cadena de texto puede ir en su lugar. En el ejemplo se ha indicado que puede ser cualquier *string* que termine con %.
- *pos*: *Part of Speech* en inglés, que viene a ser la categoría gramatical (sustantivo, adjetivo, determinante, etc).
- *sense*: Significado del término. Se señalará uno previamente creado por el módulo de desambiguación de términos de KYOTO.

II.1.2.2. Relaciones entre las variables

Una vez definidas las variables, se establece el orden y las posiciones en las que puede estar cada una de ellas. En el ejemplo, A es el pivote principal. Después de A se encuentra B. Tras B debe ir C, y por último, D debe ir después de C. B debe de ir inmediatamente después de A, es decir, no puede haber ningún término entre estas dos variables.

```
<relations>
  <root span="A"/>
  <rel span="B" pivot="A" direction="following" immediate="true" />
  <rel span="C" pivot="B" direction="following" />
  <rel span="D" pivot="C" direction="following" />
</relations>
```

Figura II.5: Definición de las relaciones entre las variables

El primer elemento siempre deberá ser la raíz *root*, y el atributo *span* indicará qué variable cumple dicha función. A partir de ahí, se irán añadiendo tantas relaciones *rel* como haga falta. Cada relación podrá tener los siguientes atributos:

- *span*: Nombre de la variable que creará una nueva relación.
- *pivot*: Nombre de la variable que establecerá la relación con *span*. Debe haber sido mencionada en algún atributo *span* previamente, ya sea en alguna relación o en la raíz.
- *direction*: En qué dirección tiene que estar la variable *span* respecto al *pivot*: *following* (después) o *preceding* (antes).
- *immediate*: Condiciona si entre el *span* y el *pivot* puede haber más términos o por el contrario si tienen que estar juntas.
- *opt*: Determina si la relación es opcional o no.

II.1.2.3. Formato del evento

La última sección determina el formato de los eventos que creará el kybot. Por ejemplo, este kybot crea un evento con dos roles por cada patrón que encaje con todas las condiciones establecidas. Este evento contendrá el lema y categoría gramatical de la variable A. Es decir, del sustantivo común. Además, se crearán dos roles por cada evento: uno para informar qué tipo de acción ha sido realizada (incrementar, decrementar), y otro para saber la cantidad, que se encuentra en la variable D.

Cada kybot generará un único tipo evento, es decir, no se generarán dos o más eventos de un único kybot. En cambio, podrán asignarse a dicho evento tantos roles como el usuario quiera, siempre y cuando no sean a la variable

```
<events>
  <event target="$A/@tid" lemma="$A/@lemma" pos="$A/@pos"/>
  <role  target="$B/@tid" rtype="accion" lemma="$B/@lemma"/>
  <role  target="$D/@tid" rtype="cantidad" lemma="$D/@lemma"/>
</events>
```

Figura II.6: Definición de los resultados que producirá el kybot

```
<?xml version="1.0" encoding="UTF-8"?>
<kybotOut>
<doc shortname="corpus_es_abrir.wsd.onto.kaf">
<event target="t92_9_es_" lemma="abrir" pos="VMIP3S0" synset="eng-30-01346003-v" rank="0.140133" profile_id="a.xml" eid="e1"/>
<role target="f" rtype="arg" lemma="arcada" pos="NCFS000" profile_id="a.xml" event="e1" rid="r1"/>
</doc>
</kybotOut>
```

Figura II.7: *Output* generado por un kybot

que actúe de evento. Tanto los eventos como los roles tendrán el siguiente formato:

- *target*: Indica la variable que se mostrará como evento o role.
- *lemma*: Mostrará el lema de la variable indicada.
- *pos*: Mostrará la categoría gramatical de la variable indicada.
- *rtype*: Sólo para los roles. Muestra la cadena de texto que el usuario indique. Su función es sólo informativa. En nuestro ejemplo hemos puesto *acción* y *cantidad* para que se sepa qué está mostrando.

La figura II.7 muestra un ejemplo de el resultado generado tras ejecutarse un kybot.

II.2. Mining module

Una vez explicados los kybots y los documentos KAF, se puede entender la funcionalidad del módulo de minería. Como hemos mencionado anteriormente, el objetivo de nuestra interfaz es que el usuario ya no tenga que interactuar directamente con este módulo, ya que puede resultar algo complejo aún teniendo los conocimientos técnicos necesarios. La figura II.8 muestra

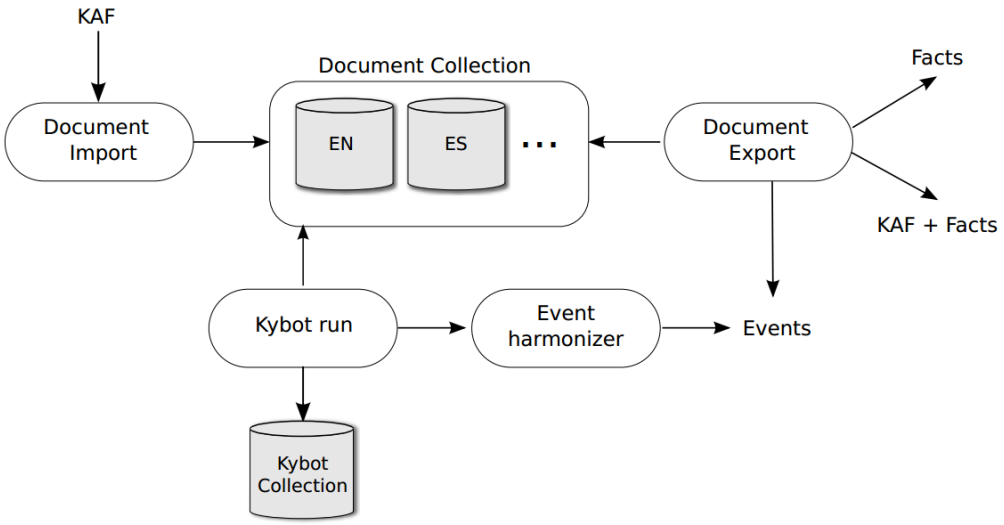


Figura II.8: Arquitectura del módulo de minería

la arquitectura del módulo, así se podrá entender mejor la explicación de su funcionamiento.

Una vez los procesadores lingüísticos han procesado los documentos de entrada al sistema, los documentos KAF resultantes son almacenados en uno de los contenedores de la BBDD Berkeley[2], una base de datos nativa en XML. Por otro lado, en otro contenedor se irán almacenando los kybots, y podrán ser ejecutados sobre los documentos KAF. Como antes se ha mencionado, el resultado de los kybots será una serie de eventos o hechos. Estos eventos se añadirán a los docuemntos KAF que correspondan, enriqueciendolos así con más información. De esta forma, los usuarios irán obteniendo los hechos más relevantes extraídos de los documentos KAF.

Como hemos apuntado en la introducción, todos estos procesos son realizados mediante comandos en el terminal de Linux, por lo que puede resultar bastante costoso para la mayoría de los usuarios. Además, las visualización de resultados en dicho terminal no resulta para nada cómoda. Otro de los inconvenientes anteriormente citados es que antes de poder ejecutar cualquiera de estos comandos, hay que comprobar que tanto la versión de la BBDD, como la de los scripts y el compilador, entre otros componentes, son compa-

[2]http://www.oracle.com/us/products/database/berkeley-db/xml/overview/index.html

tibles entre sí y que no ha habido ningún problema durante las instalación de cada una de estas herramientas.

Por tanto, queda claro que la creación de una interfaz facilitaría mucho el uso de este módulo a la mayoría de los usuario del sistema KYOTO.

III. CAPÍTULO

Documento de objetivos de proyecto

III.1. Alcance

El objetivo de este proyecto será crear una interfaz para facilitar la interacción con el módulo de minería de textos de KYOTO. Esta interfaz deberá ser implementada, como muy tarde, para principios de julio de 2013, recortando ciertas de las funcionalidades propuestas en caso de ser necesario. Para ello, se realizará una planificación con algo de margen a esa fecha límite. De esta forma, estará asegurada la consecución del proyecto antes de terminar ese plazo. Por tanto, se dará prioridad a integrar todo lo relacionado a la creación, edición, y ejecución de kybots, y a la visualización del resultado, y se dejarán como opcionales el resto de características. En resumidas cuentas, el objetivo principal es crear la interfaz lo más avanzada y funcional posible (dentro de las fechas establecidas), para que el usuario consiga los resultados con su kybot en mucho menos tiempo que sin el uso de esta interfaz.

III.1.1. Objetivos

Se han propuesto que la interfaz tenga las siguientes características:

- La interfaz servirá para visualizar tanto Kybots como KAFs.
- Los usuarios podrán acceder a ella a través de internet, usando cualquier navegador web y sin la necesidad de instalar componentes adicionales.

- Los módulos de la interfaz deben funcionar de manera asíncrona, de tal manera que el usuario no tenga que esperar a que acabe una de las operaciones para empezar con otra.

- Debe tener las mismas funcionalidades que el *Mining Module*. Para ello debe cumplir los siguientes requisitos:

  - La aplicación debe tener enlazada dos contenedores de BBDD XML Berkeley para que en ellos se almacenen los kybots y KAFs.
  - Los usuarios podrán visualizar los KAFs y kybots almacenados en dichos contenedores.
  - Podrán añadirse más kybots y KAFs a los contenedores.
  - Podrán ejecutarse kybots sobre los archivos KAF almacenados.

- Todas las funcionalidades anteriormente mencionadas deben poder realizarse de una forma rápida e intuitiva, sin que el usuario interaccione con la capa del *Mining Module*.

- Tanto los documentos KAF como los kybots deben visualizarse en un formato más fácil de entender para los usuarios.

- El usuario podrá descargarse los kybots a su ordenador.

- Al ejecutarse un kybot, el usuario verá al momento los resultados producidos. Además tendrá la opciones de verlos tal y como los generaba el *Mining Module*, o de una forma más intuitiva.

- El usuario podrá interaccionar con estos resultados: Podrá visualizar las características de cada término, añadir al kybot que esté modificando dichas características.

- Podrán crearse kybots directamente desde la interfaz, o bien se podrán subir Kybots ya creados.

- Una vez se muestre un kybot en la pantalla, el usuario podrá editarlo de tal forma que no tenga que interaccionar con el archivo XML e irá visualizando los cambios al momento y de una forma más intuitiva en la interfaz.

- Los cambios que se vayan realizando a los kybots podrán ser revertidos mediante un historial de acciones.

- Gestión de usuarios: Los usuarios tendrán que identificarse antes de usar la aplicación. De esta forma, cada usuario, en función de su rol, tendrá acceso a un determinado número de funcionalidades. Por otra parte, cada usuario verá sus kybots previamente creados/subidos.

III.2. Método de trabajo

III.2.1. Gestión

Este proyecto tiene varias particularidades. La primera es que será sido dirigido por dos personas: German Rigau y Aitor Soroa. Además, desde septiembre hasta febrero, el alumno tendrá que compaginar el proyecto con unas prácticas de 5 horas diarias que está realizando en Madrid hasta principios de marzo.

Las reuniones, debido a los factores mencionados, tendrán que ser realizadas mediante Skype en gran parte, excepto cuando sea necesario quedar en persona. Éstas se harán cuando el alumno cumpla con los objetivos marcados en las reuniones anteriores. Para la comunicación diaria las dudas que vayan surgiendo se usará el correo electrónico. Una vez terminada cada reunión, se escribirá en un documento compartido en Google Drive los puntos comentados, y los objetivos a cumplir de cara a la siguiente reunión.

Por tanto, el método de desarrollo elegido ha sido iterativo incremental, ya que es el que mejor se adapta a estas condiciones. El proyecto estará dividido en varios bloques, y cada reunión fijará los objetivos del siguiente bloque. En todas las iteraciones se llevará a cabo un proceso de trabajo similar, excepto en las que se requiera un periodo de formación previa para usar ciertas herramientas.

En cuanto a las copias de seguridad, se realizará una copia incremental periódicamente. En los primeros meses, hasta marzo, se realizará cada viernes. Después, hasta finalizar el proyecto, se realizarán los martes y viernes. La razón de esto es que en la segunda fase se invertirán más horas diarias en el proyecto, por lo que los daños por pérdida de datos serían mayores. Dichas copias se almacenarán en Dropbox y en un disco duro externo.

III.2.2. Recursos

El alumno usará su portatil para el desarrollo de la aplicación, ya que no requiere ningún tipo de *hardware* especial ni tampoco mucha potencia de procesamiento. Por otra parte, siempre que sea posible, el ordenador estará conectado a internet para comunicarse con los tutores o para buscar información relacionada con el proyecto. También se usará un disco duro externo para almacenar las copias de seguridad que se vayan realizando.

En lo referente a servicios Web, se usarán los ya mencionados Dropbox para las copias de seguridad, Gmail para la comunicación con los tutores, Drive para la gestión de actas y documentos, y Skype para las videoconferencias.

Eclipse será el entorno donde se desarrolle la aplicación. En dicho entorno se usará la API para Java de la BBDD XML Berkeley para gestionar todo lo referente a BBDD. Por otra parte, se instalará el plugin de GWT (*Google Web Toolkit*) para desarrollar la aplicación web junto con el plugin GAE (*Google App Engine*) para hospedar la aplicación en la infraestructura Google y que sea accesible desde cualquier ordenador.

El software elegido para realizar las copias de seguridad es Cobian Backup. Para realizar la memoria, se usará el sistema de composición de textos LaTex, y TeXnicCenter como editor. Para los diagramas UML, tablas, etc., se usarán las herramientas Visual Paradigm y GanttProyect.

Los tutores también han facilitado varios recursos: Por una parte, documentos en relación con los kybots y el proyecto KYOTO, como *Kyoto: a knowledge-rich approach to the interopera-ble mining of events from text* (Piek *et al.* (2013)), *Kybots, knowledge yielding robots* (Rigau (2009)), *Fact miners revised (Rigau* et al. *(2010)) y Storyboard: "to mine by example for building kybots* (German eta Eneko (2007)).

Por otra parte también proporcionaron todo lo necesario para instalar y ejecutar el módulo de minería[1] (Vossen *et al.* (2012)). Por otra parte, algunos enlaces a aplicaciones similares para intentar tener una idea inicial y saber unas pautas básicas sobre cómo desarrollar la aplicación. Entre ellos el *KAFinspector*[2] y el *Kyoto demo*[3].

[1] https://kyoto.let.vu.nl/svn/kyoto/trunk/modules/mining_module/
[2] http://weblab.iit.cnr.it/kafdebugger/
[3] http://ixa2.si.ehu.es/demokaf/demokaf.pl

III.3. Planificación

III.3.1. Estructura de Descomposición del Trabajo

La figura III.1 nos muestra los paquetes más importantes en los que se ha dividido el proyecto. El listado de actividades desglosará con un poco más de detalle los paquetes más grandes del diagrama EDT, para así poder planificar mejor todos los apartados a realizar durante las diferentes fases del proyecto.

III.3.2. Listado de actividades

- Gestión
  - Definir DOP
    - Reuniones
      - Revisar los puntos de la anterior reunión
      - Resolver dudas nuevas
      - Realizar lista de tareas para la siguiente reunión
    - Copias de seguridad
      - Copias regulares
      - Copias incrementales
      - Almacenamiento: Nube + HDD externo
    - Análisis de antecedentes
      - KYOTO
      - Interfaz similar
- Formación
  - Tecnologías necesarias para el desarrollo
    - *Google Web Toolkit*
    - *Google App Engine*
    - BBDD Berkeley XML + XQuery
  - KYOTO
- Desarrollo

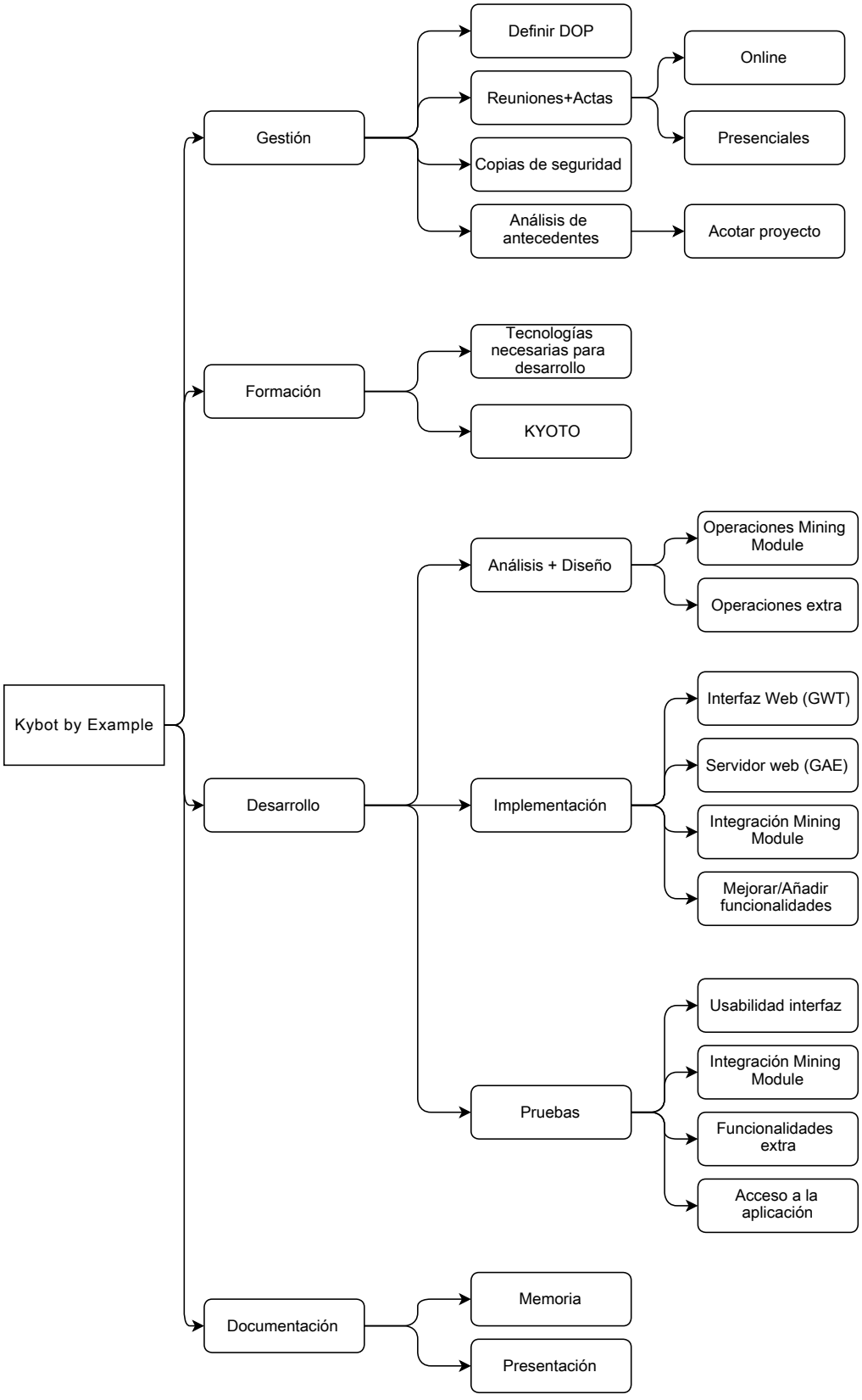


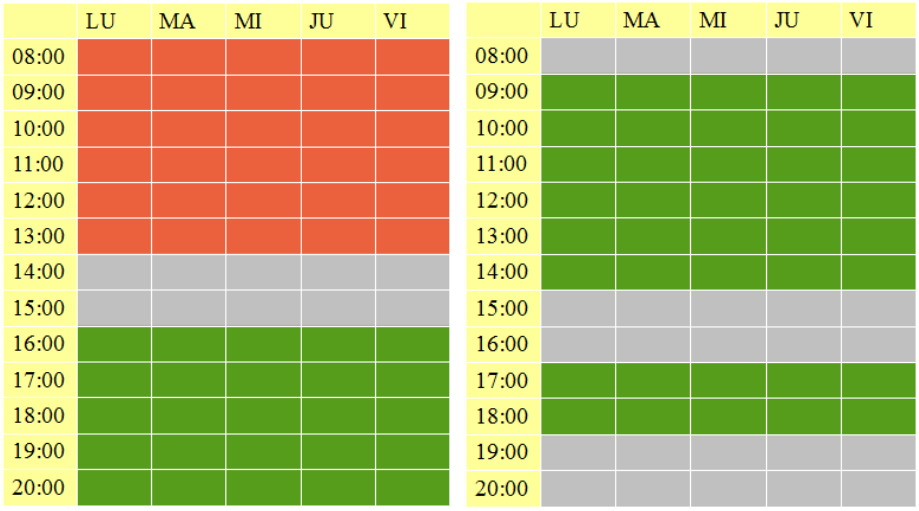
Figura III.1: Diagrama EDT

- Análisis + Diseño
  - Operaciones *Mining Module*
    - ◇ Carga, visualización, ejecución kybots
    - ◇ Visualización KAFs
  - Operaciones Extra
    - ◇ Edición avanzada kybots
    - ◇ Visualización simplificada e interactiva componentes
    - ◇ etc.
- Implementación
  - Interfaz Web (GWT)
  - Servidor Web (GAE)
  - Integración *Mining Module*
  - Mejorar/Añadir funcionalidades
- Pruebas
  - Usabilidad interfaz
  - Integración *Mining Module*
  - Funcionalidades Extra
  - Acceso a la aplicación

- Documentación
  - Memoria
  - Presentación

III.3.3. Calendario de trabajo

En los primeros 5 meses del proyecto (desde principios de septiembre hasta principios de marzo) la carga de trabajo será de unas 4 horas diarias de lunes a viernes (figura III.2(a)), debido a que se compaginará con las prácticas anteriormente mencionadas. Durante este periodo, se estima que habrá unos 12 días en los que no se trabajará, ya sea por días festivos, enfermedad, o cualquier otro contratiempo. Tras esto, la dedicación al proyecto será plena, y el volumen de horas subirá a las 7 diarias desde marzo hasta finalizar el proyecto (figura III.2(b)). Durante este periodo se estima que habrá alrededor de 8 días en los que no se dedicará tiempo al proyecto.



	LU	MA	MI	JU	VI
08:00					
09:00					
10:00					
11:00					
12:00					
13:00					
14:00					
15:00					
16:00					
17:00					
18:00					
19:00					
20:00					

(a) Horario de trabajo con prácticas (Septiembre-Marzo)

	LU	MA	MI	JU	VI
08:00					
09:00					
10:00					
11:00					
12:00					
13:00					
14:00					
15:00					
16:00					
17:00					
18:00					
19:00					
20:00					

(b) Horario de trabajo sin prácticas (Marzo-fin del proyecto)

Figura III.2: Horarios de trabajo establecidos

III.3.4. Estimaciones globales

Tras la descomposición de las distintas tareas del proyecto, y tras planificar el volumen diario de horas que se invertirá en el proyecto, podemos determinar la distribución y el volumen de horas de los distintos paquetes, lo cual puede verse en la figura V.4.

Una vez se ha estimado la cantidad de horas de cada tarea y por tanto se sabe la duración aproximada del proyecto, se ha relizado el diagrama de Gantt de la figura III.4. Puede apreciarse que se ha planificado que el proyecto finalizará a finales de abril o principios de mayo. Ya que la fecha límite para terminar el proyecto es a principios de julio, tenemos un margen de unos dos meses para posibles contratiempos y percances.

III.4. Plan de contingencia de los riesgos

A continuación se exponen los riesgos más significativos que afecten al desarrollo del proyecto. Para cada uno de ellos, se analizarán las causas, sus consecuencias, y qué medidas se tomarán para hacerlos frente.

- **Pérdida o destrucción de datos *(Prob. media \ Gravedad alta)***

Categoría	Tarea	Horas est.
Gestión	Definir DOP	20
	Reuniones + Actas	25
	Copias de seguridad	5
	Análisis antecedentes	25
Formación	KYOTO	30
	Google Web Toolkit	25
	Google App Engine	5
	Berkeley DB + XQuery	30
Desarrollo		
Análisis	Operaciones módulo minería	6
	Operaciones Nuevas	15
Diseño	Operaciones módulo minería	15
	Operaciones Nuevas	25
Implementación	Interfaz Web (GWT)	50
	Servidor Web (GAE)	15
	Integración módulo minería	40
	Mejorar y añadir funcionalidades	200
Pruebas	Usabilidad interfaz	15
	Integración módulo minería	15
	Acceso a la app vía Web	5
	Operaciones nuevas	15
Documentación	Realizar memoria	90
	Preparar diapositivas/presentación	15
	TOTAL	**686**

Figura III.3: Estimación de horas para cada paquete

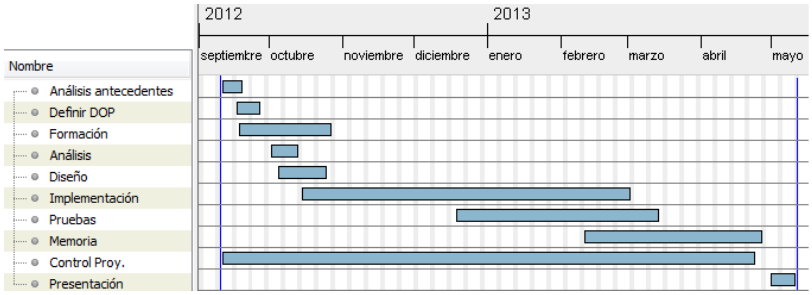


Figura III.4: Diagrama de Gantt planificado

- Causas:
  - Destrucción o corrupción de datos debido a viruses, error humano o por problemas físicos con el soporte que almacena los datos en el ordenador.
  - Extravío del equipo de trabajo.
- Consecuencias:
  - Pérdida de la información del proyecto.
  - No cumplir con los objetivos de tiempo planificados.
- Medidas a tomar:
  - Realización de copias de seguridad incrementales periódicas, las cuales se almacenarán en la nube y en un disco duro externo.
  - Proteger el ordenador frente a virus, troyanos, etc.
  - Tener un segundo equipo disponible.

- **No cumplir los tiempos estimados en la planificación *(Prob. media \ Gravedad media)***
  - Causas:
    - Realizar una mala planificación.
    - No cumplir con la planificación.
    - Contratiempos inesperados.
  - Consecuencias:
    - No conseguir los objetivos en el tiempo planificado.

    - Tener que replanificar de nuevo las tareas restantes.
    - Necesidad de invertir horas extra para cumplir los plazos.
  - Medidas a tomar:
    - Planificar correctamente y dejando margen para posibles contratiempos.
    - Seguir con disciplina los horarios planificados.
    - En caso de preveer que no se van a cumplir los tiempos estimados, invertir más horas de las planificadas.

- **Indisposición temporal de alguno de los participantes del proyecto *(Prob. baja \ Gravedad baja)***
  - Causas:
    - Enfermedad de cualquiera de los participantes.
    - Compromisos tales como reuniones, conferencias, etc. por parte de los tutores.
  - Consecuencias:
    - Retrasos para cumplir los objetivos planificados.
    - Imposibilidad de comunicarse con alguno de los participantes.
  - Medidas a tomar:
    - Tener tareas alternativas para realizar hasta que la persona vuelve a estar disponible.
    - Tener vías alternativas de comunicación, tales como *e-mail*, teléfono, etc.

- **Problemas con *hardware* o software *(Prob. baja \ Gravedad baja)***
  - Causas:
    - *Bugs* en el software que se usará para el desarrollo.
    - Incompatibilidad entre distintos componentes de desarrollo.
    - *Hardware* poco potente o defectuoso.
  - Consecuencias:
    - Retrasos para cumplir los objetivos planificados.

    - Volver a diseñar la aplicación en caso de que cierto software no sea usable, o no sea compatible con otros componentes.
  - Medidas a tomar:
    - En cuanto al *hardware*, tener un ordenador con pocos años de antigüedad para poder soportar los requerimientos del software que se usará, y tener un ordenador de repuesto en caso que se averíe el principal.
    - En cuanto al software, hacer un análisis al comienzo del desarrollo para ver que los componentes que se usarán serán compatibles entre sí. Por otra parte tener actualizados los componentes para así tener las versiones más sólidas y estables.

- **Problemas de desarrollo de la aplicación *(Prob. media \ Gravedad media)***
  - Causas:
    - Realizar un mal análisis.
    - Realizar un mal diseño.
    - Falta de formación en cierta tecnología requerida para desarrollar la aplicación.
  - Consecuencias:
    - Mala o ineficiente implementación.
    - No cumplir a tiempo con los objetivos establecidos.
    - Necesidad de volver a realizar otra planificación.
  - Medidas a tomar:
    - Tener en cuenta este problema a la hora de planificar, y estimar un poco más del tiempo que se espera.
    - Tener una buena formación antes de usar cualquier tecnología que el alumno no domine.
    - Invertir el tiempo que sea necesario en realizar un buen análisis y diseño.

III.4.1. Factibilidad

Si el tiempo no fuese una limitación, todas las características mencionadas en el alcance podrían ser implementadas sin problema. Como se ha

mencionado arriba, se ha dejado un colchón de dos meses para cualquier imprevisto que surja durante el desarrollo del proyecto. También hay que tener en cuenta que el alumno estará varios meses con el horario reducido de 4 horas diarias, por lo que la percepción del tiempo disponible puede ser engañosa. Por tanto, las siguientes funcionalidades han sido excluidas de la planificación y sólo serán implementadas si se dispone de tiempo suficiente:

- Gestión de usuarios: Se requeriría mucho tiempo para desarrollar y probar esta característica. Además, de momento no sería esencial integrar esta funcionalidad, ya que tampoco hay muchas operaciones críticas como la de borrar las BBDD o similares. Por tanto, se implementará cuando sea necesaria un control sobre qué acciones puede realizar cada usuario.

- Historial de acciones: Al igual que la anterior, a pesar de ser una opción interesante, no necesita ser añadida urgentemente a la interfaz.

- Mostrar e interaccionar con características avanzadas de los KAFs: Una vez terminada la interfaz con las operaciones más básicas e importantes, tendría que volver a hacerse un análisis para ver qué nuevas funcionalidades avanzadas podrían integrarse.

- Facilitar aún más el uso de la interfaz: Hacer la interfaz todavía más fácil de usar. Por ejemplo, incluyendo acciones como *drag-and-drop* y similares. Por otra parte, podría hacerse un estudio para implementar componentes que mejoren la accesibilidad para personas con problemas de visión o similares.

De esta forma, nos aseguramos de terminar a tiempo las funcionalidades más importantes de la interfaz, y evitamos el estancamiento en otras que tal vez no sean de vital importancia para los usuarios de KYOTO.

IV. CAPÍTULO

Desarrollo de la aplicación

IV.1. Captura de requisitos

A continuación se muestran los requsitos principales que deberá cumplir la interfaz.

- Funciones de consultas, creación y edición, etc. de los datos
  - Consulta de kybots y KAFs
  - Consulta de resultados generados por kybots en formato típico o formato fácil
  - Consulta de atributos de los KAFs
  - Creación de kybots mediante subida de fichero XML
  - Creación de kybots en la propia interfaz
  - Edición de kybots usando atributos de los resultados previos
  - Edición de kybots introduciendo datos manualmente
  - Descarga de kybots
- Funciones de interacción con otros sistemas
  - Integración con el módulo de minería (BBDD Berkeley XML + Scripts)

    - Integración con contenedor de servlets Tomcat

- Funciones de rendimiento

    - Uso de AJAX para agilizar la interacción

- Funciones de comunicación

    - Accesible mediante navegador web con Internet

IV.1.1. Modelo de dominio

La figura IV.1 muestra el modelo de dominio de la aplicación.

Los usuarios interactuarán con la interfaz web, la cuál se encargará de realizar todos los servicios disponibles del sistema.

Por un lado, la interfaz mostrará el kybot con el que esté trabajando el usuario. Este kybot estará compuesto por varios atributos, que contienen información lingüística. Estos atributos podrán editarse desde la interfaz.

En cualquier momento se podrá ejecutar el kybot sobre los documentos KAF almacenados en el sistema, y esto producirá resultados. Estos resultados son los eventos generados por los kybots, que contendrán fragmentos de los KAF que hayan cumplido las restricciones del kybot. La interfaz mostrará estos resultados, junto con su información lingüística, y ésta última podrá usarse para editar los atributos del kybot.

IV.1.2. Definición de la interfaz de usuario

Una vez se saben las funciones principales que deberá tener la aplicacíon, se puede esbozar un primer esquema, el cual se muestra en la figura IV.2, de los módulos que tendrá la interfaz. Consistirá en una única ventana, la cual tendrá 3 módulos principales.

Por un lado el panel donde se muestre el kybot que esté visualizando el usuario, en el que se podrán realizar las tareas que indican los botones, y además editar el kybot en sí. En la figura está identificado como *OutputKP*.

Por otro lado, habrá otro panel en el que se mostrarán los resultados producidos por el kybot. Como puede verse, también se dará la opción de elegir en qué formato mostrar los resultados. En la figura aparece como *WfPanel*.

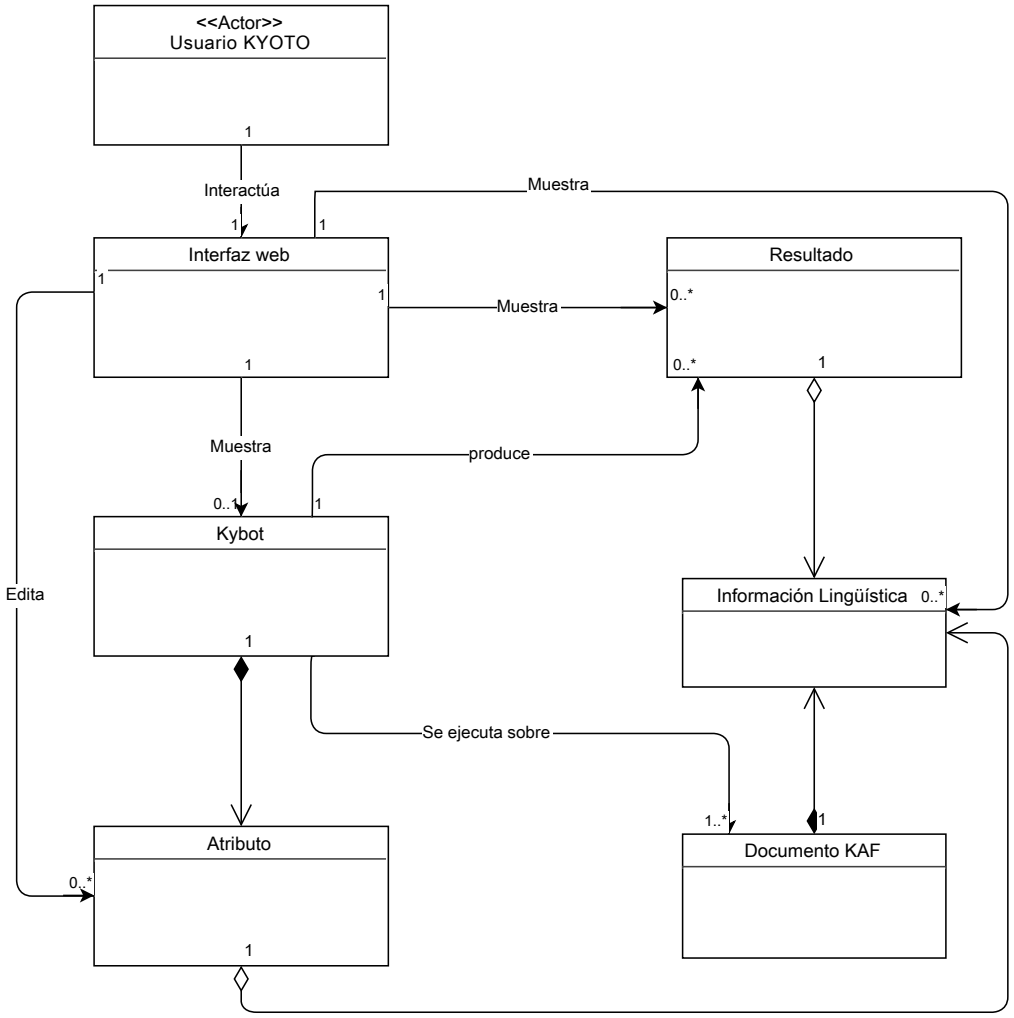


Figura IV.1: Modelo dominio

Por último, el panel lateral servirá para mostrar la información de los términos que el usuario seleccione en el panel de resultados. En la figura aparece como *Selection*.

IV.1.3. Modelo de casos de uso

La figura IV.3 muestra las operaciones que podrá realizar el usuario. Por el momento, al no haber gestión de cuentas, no hay ningún administrador, por lo que el único rol que hay es el de usuario de KYOTO. A continuación se explica en detalle cada operación.

- Crear kybot: Creación de un kybot desde la interfaz. Esto se realizará en la módulo *OutputKP* mostrado en la figura IV.2. Una vez el usuario haya pulsado el botón de crear un kybot, deberá asignarle un nombre y definir su primera variable. Para ello, introducirá un nombre que la identifique y tendrá que definir sus atributos: lema, *part of speech*, o ambos. Una vez se haya suministrado toda esa información se mostrará en pantalla empleando el caso de uso *Mostrar kybot*, que se explicará más adelante.

- Cargar kybot: En vez de crearlo desde la interfaz, se podra cargar en el sistema un archivo XML que contenga un perfil kybot. También se realizará desde el módulo *OutputKP*. Una vez el usuario haya pulsado el botón de cargar un kybot, el sistema parseará el archivo XML, y si está construido correctamente, lo mostrará en pantalla con el caso de uso *Mostrar Kybot*.

- Descargar kybot: Cuando el usuario pulse el botón de descargar kybot del panel *OutputKP*, comenzará la descarga del archivo XML del kybot que se esté mostrando en pantalla. Por lo tanto, se tendrá que haber subido o creado un kybot antes de poder realizar esta tarea.

- Ejecutar kybot: Cuando el usuario pulse el botón de procesar el kybot del panel *OutputKP*, se procederá a la ejecución del kybot que se muestra en pantalla usando los scripts del módulo de minería. Estos, se encargarán de acceder a la BBDD y realizar las tareas necesarias para que se realice la operación.

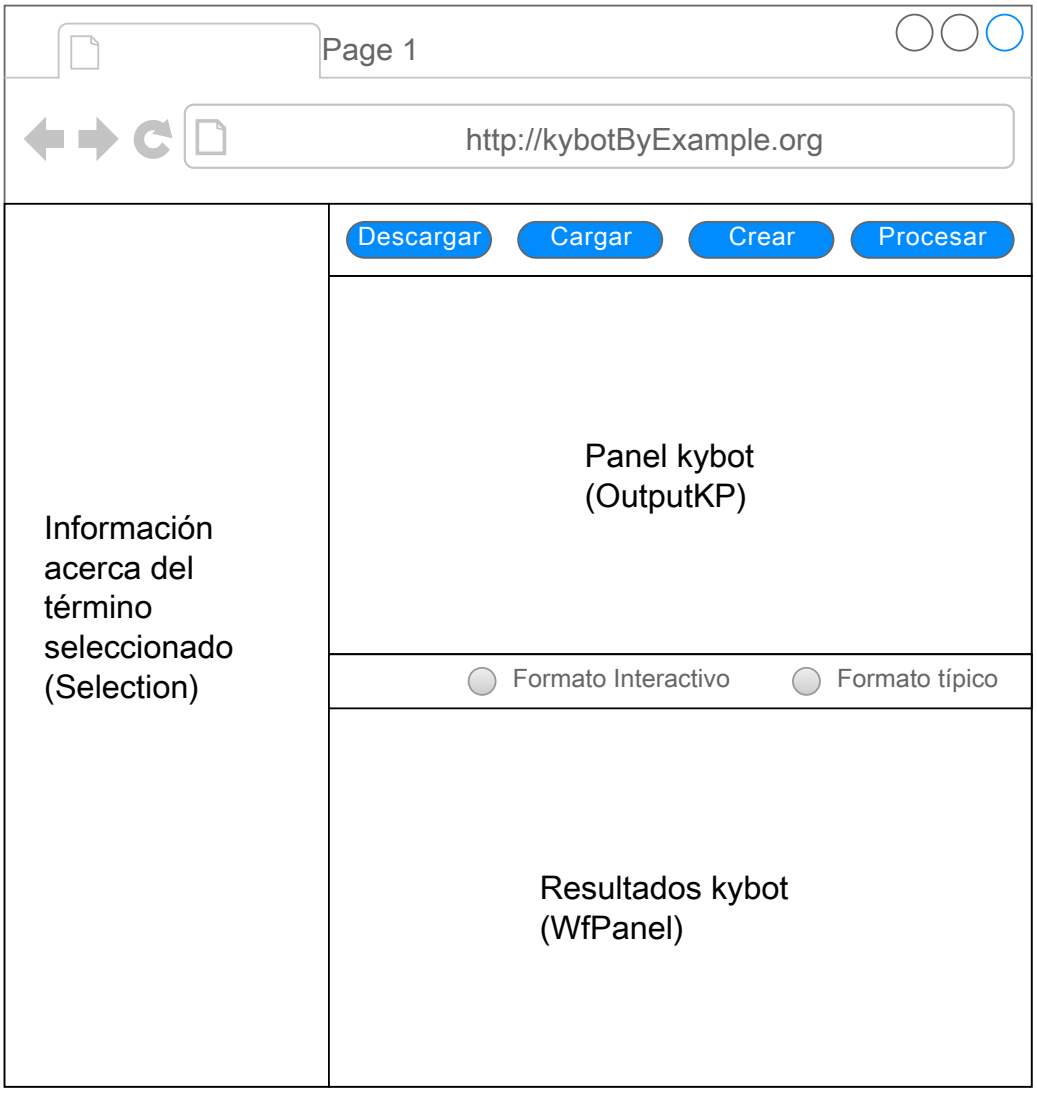


Figura IV.2: Esquema de la interfaz de usuario.

- Ver *output* del kybot: Visualización del resultado en el módulo *WfPanel* de la interfaz, inmediatamente después del caso de uso *Ejecutar kybot*. Habrá dos opciones de visualización del *output*.

  - Formato típico: Si el usuario ha elegido la opción de *formato típico* en el módulo *WfPanel*, el resultado se mostrará como texto plano y sin posibilidad de interaccionar con él, como hasta ahora se ha realizado en el módulo de minería. El usuario tan solo podrá ver en pantalla el resultado generado por el kybot.
  - Formato interactivo: En cambio, si el usuario ha elegido la opción de *formato interactivo* en el módulo *WfPanel*, el sistema, a partir del resultado típico, creará un *output* alternativo que será mucho más fácil de entender para los usuarios.

    Además, éste será interactivo. Es decir, el usuario podrá seleccionar cualquiera de los términos del resultado obtenido para ver sus atributos en el módulo *Selection* de la interfaz. Aquí, el usuario podrá ver el identificador del término seleccionado, su *pos*, lema, y referencias externas. El usuario entonces podrá añadir alguna de esas características al kybot que se esté visualizando. Esto se explicará en el siguiente caso de uso.

- Editar kybot: Posibilidad de cambiar/añadir/borrar atributos al kybot que se esté mostrando en pantalla. A continuación se muestran las dos formas de editar el kybot:

  - Editar a partir del *output* interactivo: Como se ha explicado anteriormente, el usuario podrá editar el kybot que se muestra en pantalla a partir de los componentes del *output* interactivo. Para ello, se tendrá que seleccionar del *output* la característica deseada, y luego seleccionar la variable del kybot que se desee editar. La interfaz, entonces, mostrará la siguiente opción para editar el kybot:

    - *Añadir nueva variable delante/detrás de la variable seleccionada*: El usuario podrá añadir una nueva variable con el atributo que haya seleccionado del *output* interactivo. Ésta podrá ir delante o detrás de la variable seleccionada, y también se deberá especificar el atributo *immediate*, es decir, si se permitirá introducir variables entre la seleccionada del kybot y la

que se creará. Para terminar la operación se deberá especificar el nombre la nueva variable.

- Editar manualmente: Cualquier tipo de edición que no requiera interaccionar con el *output* interactivo. El usuario simplemente seleccionará la variable del kybot que desee modificar, y la interfaz le dará las siguientes posibilidades de edición:
  - *Modificar información de la variable seleccionada*: Se dará la opción de modificar las características de la variable seleccionada, como el *pos*, el lema y la referencia externa.
  - *Borrar la variable seleccionada*: Se borrará la variable, junto con todas las que tengan alguna dependencia con ésta.
  - *Añadir nueva variable delante/detrás de la variable seleccionada*: Muy similar a la opción ofrecida para el *output* interactivo. La única diferencia será que en vez de tener una característica escogida de antemano, el usuario tendrá que introducirla manualmente.
  - *Seleccionar la variable como generadora del evento del kybot*: Si la variable seleccionada cumple los requisitos (no tener el atributo *role* ni *event*), se dará la opción de convertir la variable seleccionada como generadora del evento del kybot. Como se explicó anteriormente, ya que sólo puede haber una variable que genere los eventos, la variable que tuviese este atributo lo perderá cuando se le asigne a ésta.
  - *Asignar un role a la variable*: Al igual que con la anterior opción, si la variable no tiene el atributo *role* ni *event* de antemano, se *dará* la opción de asignar un *role* a la variable seleccionada.
  - *Borrar el role de la variable seleccionada*: Si la variable seleccionada tenía un *role* previamente asignado, se mostrará esta opción para eliminar dicha característica.

- Mostrar kybot: Proceso que mostrará el kybot con el que se esté trabajando en pantalla. Esto se realizará después de los casos de uso de *crear kybot*, *cargar kybot* o *editar kybot*, los cuales han sido explicados en los anteriores puntos. Cada vez que realiza una de estas tres operaciones se realizará este proceso, ya sea para visualizar por primera vez el kybot en pantalla, o para refrescarlo en caso de haber sido modificado.

En cualquier caso, la interfaz analizará el kybot con el que se esté trabajando, y creará una estructura similar a la de un árbol para visualizarlo en el módulo *OutputKP*. Cada una de las variables del kybot será un elemento del árbol, donde la raíz del árbol será la variable *root* del kybot. A partir de ahí, los hijos de cada variable serán los que tengan una dependencia *following* o *preceding* con ella. Por lo tanto, viendo la propia estructura del árbol se sabrán las dependencias que hay entre todas las variables.

Cada elemento del árbol, es decir, cada variable del kybot, mostrará lo siguiente en pantalla:

- El tipo de dependencia que tiene respecto a su padre mediante un símbolo: *following* con el símbolo $->$ o *preceding* con el símbolo $<-$. En caso de ser *Immediate* se mostrarán como $-->>$ y $<<--$ respectivamente. Para terminar, se mostrará la palabra *Root* en caso de que el elemento sea la raíz del kybot.
- El nombre de la variable, junto con los atributos (lema, *pos* y/o referencias externas) que tenga.
- En caso de ser la variable que genera el evento, el elemento del árbol estará rodeado de un marco negro.
- En caso de tener un *role* asignado, el elemento del árbol estará con un fondo amarillo.

Todo esto podrá entenderse más fácilmente cuando se vea un ejemplo real en la sección IV.4.1.1. de la implementación.

IV.2. Análisis

IV.2.1. Arquitectura general

En esta sección se mostrará y explicará la arquitectura global de la aplicación la cual se muestra en la figura IV.4.

La aplicación se basará en la arquitectura *cliente-servidor*. En la parte del cliente estará la interfaz web con la que interactuará el usuario. En el servidor estará el módulo de minería que procesará las peticiones que vaya recibiendo. Para recibir estas peticiones, se implementarán unos servlets, que

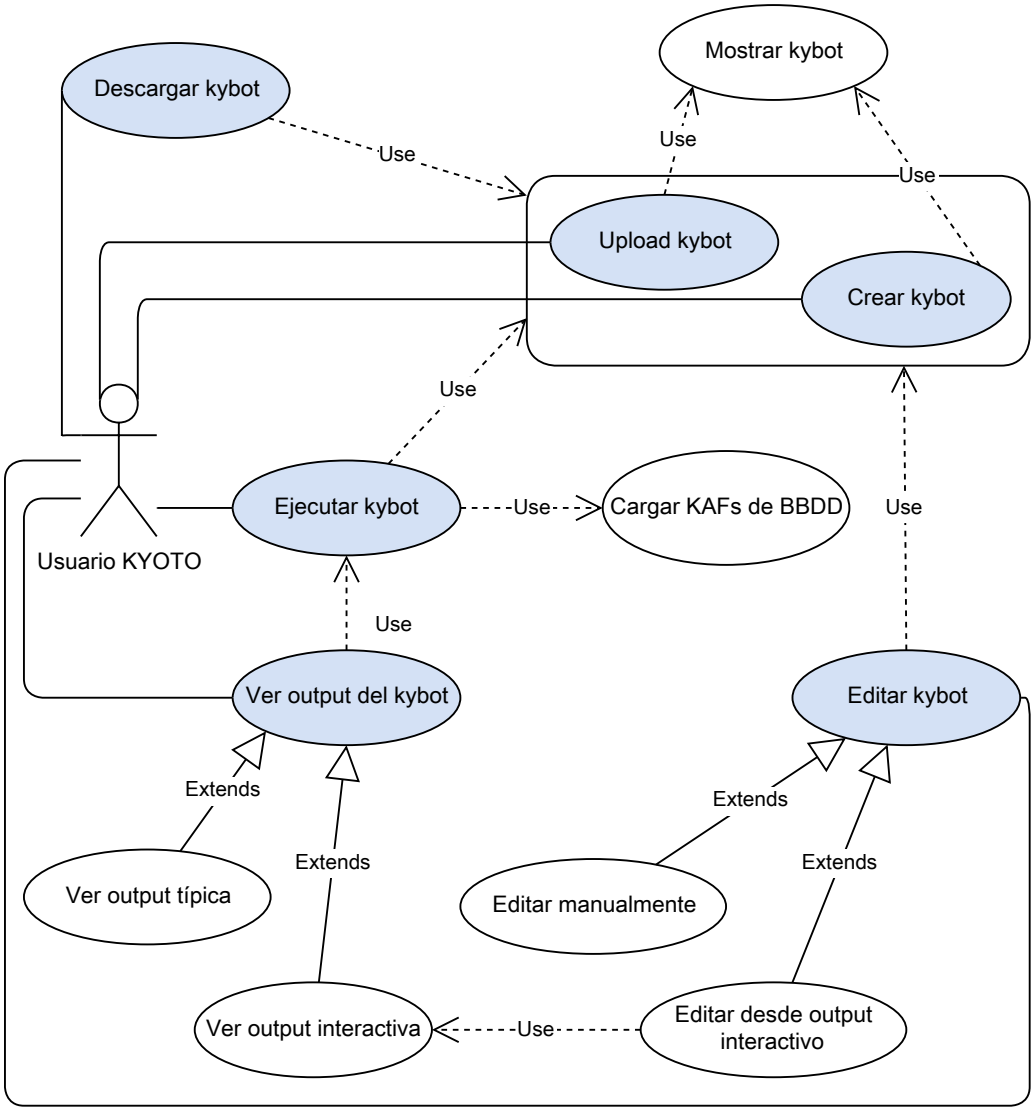


Figura IV.3: Diagrama general de casos de uso

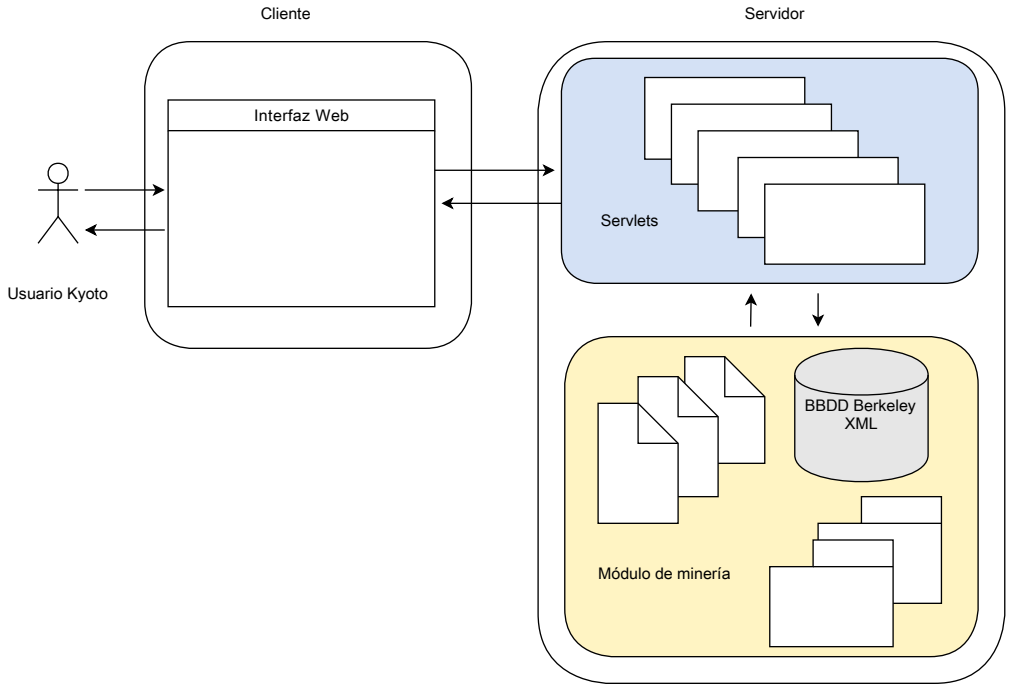


Figura IV.4: Arquitectura general de la aplicación

son programas Java que corren en la parte del servidor y sirven para recibir y responder las peticiones que se envían desde el cliente. Por tanto, en este caso se usarán para gestionar todas las tareas que el usuario vaya envíando para la edición, visualización y creación de kybots.

IV.2.2. Elección tecnológica

Una vez se sabe la arquitectura general, en esta sección se mostrarán qué herramientas y tecnologías se han utilizado para crear la aplicación. En los casos que se haya cambiado alguna herramienta durante el transcurso del proyecto, se indicarán las razones y las limitaciones técnicas que tenía la elección inicial.

IV.2.2.1. Plugin Google Web Toolkit (GWT) en Eclipse

Dos de las características que debía tener la interfaz es que se pudiese acceder vía navegador web, y que se pudiese interaccionar con cada uno de

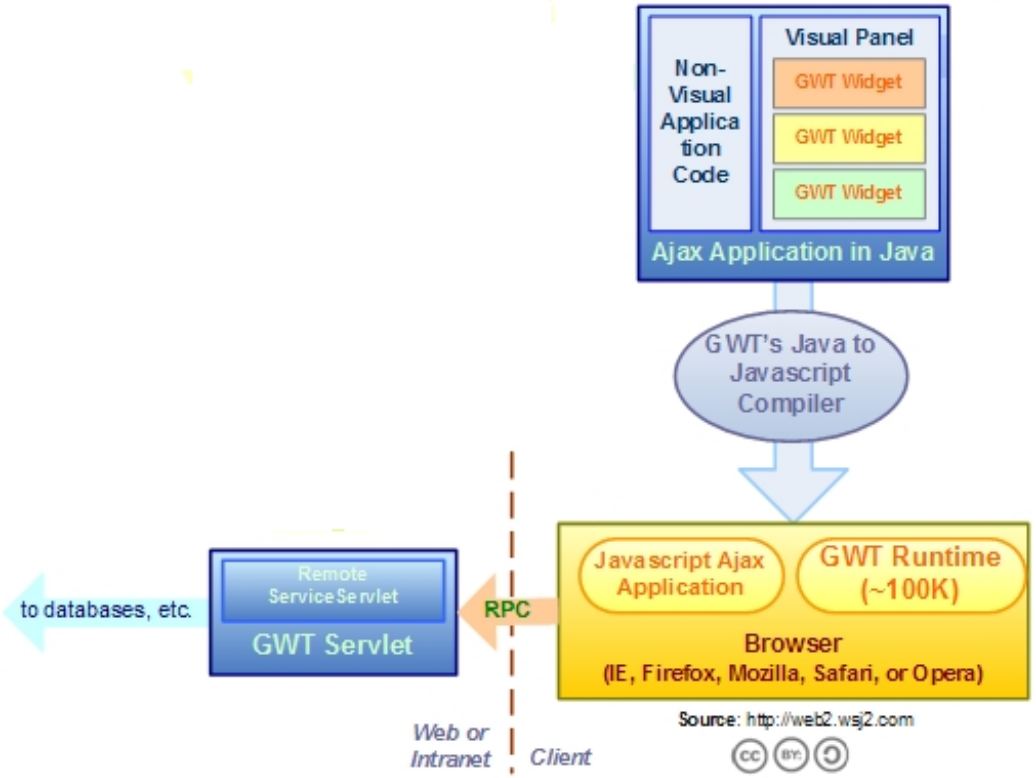


Figura IV.5: Arquitectura de GWT

los componentes de la aplicación de manera independiente, es decir, que el usuario no tuviese que esperar a terminar el proceso de un componente para interaccionar con otro.

Por esas razones se ha escogido *Google Web Toolkit*: Una herramienta para desarrollar aplicaciones web AJAX en Java. La imagen IV.5 muestra la arquitectura general de GWT. Uno de los componentes principales es el compilador GWT: Gracias a éste, se simplifican mucho las cosas, ya que el programador solo tendrá que escribir código Java, y el plugin GWT se encargará de transformarlo a HTML y Javascript, agilizando mucho el proceso.
Por otra parte AJAX permitirá la comunicación asíncrona con los servlets, pudiendo interactuar con los componentes de la aplicación individualmente sin tener que recargar toda la página una y otra vez.

Se ha elegido Eclipse como entorno de desarrollo, ya que existe un plugin

específico de GWT para éste, por lo que la integración de las dos herramientas será perfecta.

IV.2.2.2. Apache Tomcat en sustitución de Google App Engine

En un principio se había elegido *Google App Engine* como herramienta para alojar los servlets creados. Se había escogido esta herramienta ya que venía integrada con GWT, y el alumno tenía experiencia previa con ella, por lo que no hubo dudas a la hora de escoger una.
Pero durante el desarrollo, llegado el punto de integrar los scripts en Perl del módulo de minería en nuestros servlets, se pudo comprobar que GAE, por razones de seguridad, no permitía ejecutar dichos scripts. Tras pasar unas cuantas horas buscando la razón del problema y sus posibles soluciones, se optó por cambiar de herramienta para realizar la tarea. Tras analizar unos cuantos contenedores de servlets, finalmente se escogió Apache Tomcat, ya que era el más simple y el que mejor conocía el alumno para los requerimientos del proyecto.

IV.2.2.3. Berkeley DB XML

Berkeley DB XML es la base de datos utilizada en el módulo de minería para almacenar los KAFs y kybots, así que usaremos la misma para este cometido. Es una base de datos nativa en XML. Por otro lado su instalación es muy sencilla, ya que se trata de una librería. Otro dato a tener en cuenta es que en este tipo de base de datos los ficheros XML que queramos almacenar se guardan en simples archivos llamados contenedores, por lo que es muy fácil su manejo. Otro factor positivo es que tiene API para Java. Por lo tanto, esta base de datos se adapta muy bien a las necesidades de nuestro proyecto.

IV.2.2.4. Mining Module

Es el núcleo de la aplicación. Su estructura es muy simple: Consiste en la base de datos Berkeley y unos cuantos scripts en Perl y archivos de configuración que hacen funcionar las operaciones del módulo de minería.

IV.2.3. Comunicación entre los componentes del sistema

Llegados a este punto en el que se conocen todos los componentes del sistema, se podrá definir completamente el esquema *cliente-servidor* de la

figura IV.4. Por tanto, en esta sección se explicará cómo se van a comunicar los distintos componentes y finalmente se mostrará la arquitectura definitiva de la aplicación.

IV.2.3.1. Comunicación entre los usuarios y el sistema

Lo primero de todo será definir en la parte del cliente cómo se comunicarán los usuarios con el sistema. Para ello se creará una interfaz web que será con lo que interactúe el usuario durante todo el ciclo de uso de la aplicación. Esta interfaz se creará usando el framework GWT, la cual será capaz de comunicarse con el servidor, donde se realizarán todas las operaciones. La figura IV.5 muestra la composición de la interfaz y la forma en la que se comunicará con el servidor.

IV.2.3.2. Comunicación entre el cliente web y el servidor

Desde la interfaz web, por tanto, habrá que comunicarse con el resto de componentes del sistema, los cuales se encuentran en el servidor. Para esto, por una parte se usarán las *Remote Procedure Calls* (Llamadas a servicios remotos) de GWT, similar a los servlets Java. Esta infraestructura permitirá la comunicación entre el cliente web y el servidor para intercambiarse objetos Java mediante HTTP. La figura IV.5 muestra que la comunicación entre cliente y servidor se realiza mediante RPC por defecto. Más adelante, en la sección IV.4.1.1 de implementación de estos servicios remotos, se analizará en profundidad su funcionamiento.

No obstante, habrá situaciones en las que no puedan usarse estas *Remote Procedure Calls* debido a ciertas limitaciones. Cuando ocurra eso, se usarán servlets Java, los cuales se encargarán de recibir petiones del cliente web, procesarlas y mandarlas de vuelta. La figura IV.6 muestra la el funcionamiento de los servlets.

IV.2.3.3. Comunicación entre los servlets con el módulo de minería

Los servlets, aparte de comunicarse con el usuario, también necesitarán una forma de hacerlo con el módulo de minería para realizar las operaciones que envíen los usuarios.

Por una parte, los servlets usarán los scripts del módulo de minería para utilizar las funcionalidades que ya existían previamente. La API de Java

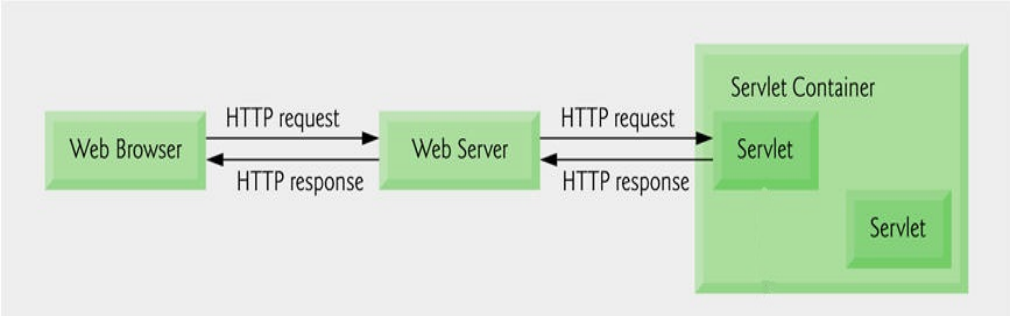


Figura IV.6: Funcionamiento de los servlets

Runtime posibilitará ejecutar esos scripts desde el código Java. Más adelante, en la sección IV.4.1.1 se mostrará el proceso para poder ejecutar los scripts.

Por otra parte, para añadir las nuevas funcionalidades a nuestra aplicación, habrá que acceder directamente a la BBDD Berkeley XML. Para ello, se usará la API para Java de Berkeley. En la figura IV.7 puede verse cómo se comunica nuestra aplicación con la BBDD mediante la API. Una vez se hace la conexión, se obtendrán los datos deseados mediante consultas en *XQuery*. También podrá verse en más profundidad en la sección IV.4.1.1.

Una vez los servlets hayan realizado las tareas oportunas y reciban los datos requeridos del mining module, se enviarán al cliente web para que los usuarios reciban los resultados.

Para terminar, hay que indicar que estos componentes de la parte del servidor estarán albergados en el contenedor de servlets Apache Tomcat. Finalmente, la figura IV.8 muestra la arquitectura final que tendrá el sistema.

IV.3. Diseño

IV.3.1. Base de datos

El único tipo de elemento que se almacena en los contenedores de Berkeley son los archivos XML. Estos serán KAF o kybots, los cuales tendrán su respectivo contenedor para su almacenamiento. Por tanto, la BBDD estará formada por dos contenedores en los que se almacenarán KAF y kybots en formato XML. Debido a la simplicidad de la estructura, no se ha realizado ningún diagrama. En las secciones II.1.1 y II.1.2 de este documento se ha

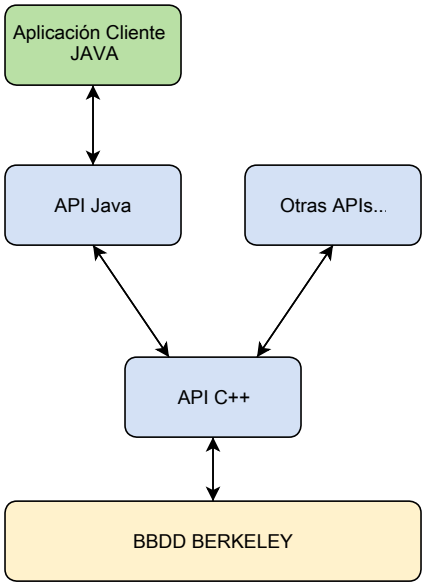


Figura IV.7: APIs en Berkeley

detallado la estructura de cada uno de estos elementos.

IV.3.2. Diagrama de clases

El diagrama de clases mostrará las clases que componen la aplicación y la relación entre ellas. Ya que sería muy complicado visualizar el esquema general en un sólo diagrama, se ha dividido en tres paquetes: las clases relacionadas con los servicios remotos y BBDD, las relacionadas con las vistas de la interfaz, y las de los objetos que se mandan en la comunicación entre servidor y cliente.

IV.3.2.1. Servicios remotos y BBDD Berkeley

En el primer grupo se analizarán las relaciones de las clases que componen los servlets Java y las *Remote Procedure Calls* de GWT, es decir, las clases implicadas en los procesos de comunicación entre cliente y servidor. No se entrará en detalles sobre cómo implementarlos, ya que eso irá en la sección IV.4.1.1. En este grupo también se muestran las clases que se relacionan con la BBDD Berkeley.

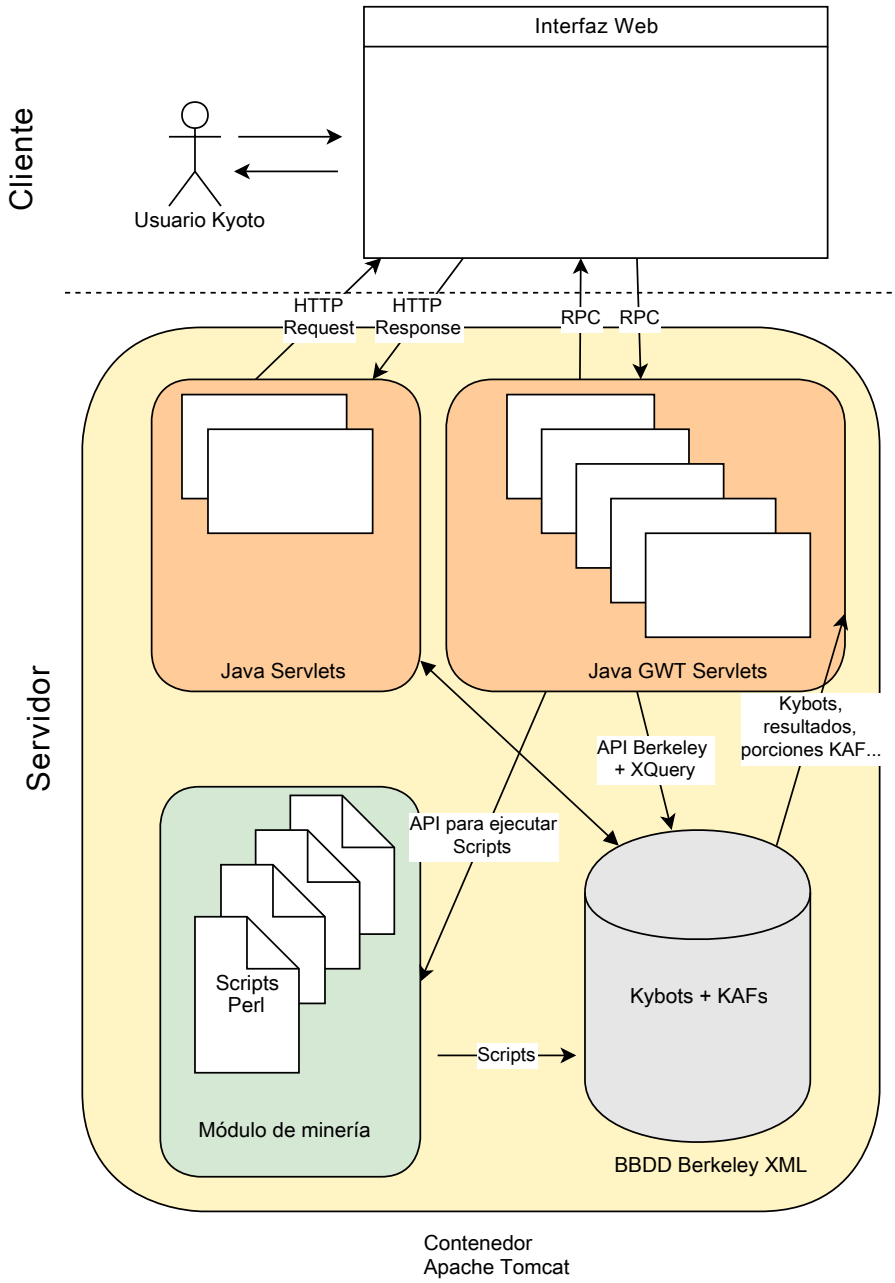


Figura IV.8: Arquitectura global del sistema

La figura IV.9 muestra un esquema general de estas clases. Por un lado, se han implementado cuatro *Remote Procedure Calls* las cuales necesitan implementar la clase *RemoteService*. Estos serán los cuatro servicios creados para interaccionar tanto con la BBDD, kybots y KAFs. Se explicarán en más detalle en las siguientes imágenes. Por otra parte, las dos clases de abajo, las cuales heredan *HttpServlet*, son los servlets Java que se han tenido que añadir para los servicios de descarga y subida de kybots al sistema. De ser posible, éstos también se hubiesen definido como *Remote Procedure Calls* para homogeneizar los componentes, pero debido a limitaciones técnicas no era posible y se ha tenido que recurrir a servlets Java.

La figura IV.10 muestra las clases relacionadas con la BBDD Berkeley. Éstas deberán instanciarse y configurarse en los servicios que necesiten acceso a los kybots o KAFs, como se verá más adelante.

Desde la figura IV.11 hasta la figura IV.14 se terminan de explicar las clases mostradas la figura IV.9:

En la figura IV.11 se muestra el servicio remoto para buscar términos en los documentos KAF almacenados en BBDD. Será necesario indicar el *tid* (identificador del término), y especificar en qué documento quiere realizarse la búsqueda. Por tanto, será necesario usar las clases relacionadas con la BBDD Berkeley, y tener acceso a los contenedores de la BBDD para realizar la operación.

En la figura IV.12 se muestra el servicio remoto para crear guardar o actualizar kybots en la BBDD. Para ello será necesario darle el objeto que contenga el kybot, y el servicio creará un documento XML y lo guardará o actualizará, si ya existía previamente en la BBDD. Por tanto, también será necesario usar las clases de Berkeley, y tener acceso a los contenedores de la BBDD para realizar la búsqueda.

En la figura IV.13 se muestra el servicio remoto de obtención de kybots de BBDD. Se le tendrá que indicar el nombre del kybot, y éste lo buscará. Una vez encontrado, éste se parseará para así crear un objeto de kybot, y finalmente lo devolverá. Aparte de necesitar las clases de Berkeley y los contendores, también será necesaria la clase *KProfile* para la creación del kybot.

En la figura IV.14 se muestra el servicio remoto para procesar un kybot. Se le tendrá que indicar el nombre del kybot, y el servicio ejecutará un script en Perl en el que se obtendrán los resultados producidos por el kybot, los cuales serán una lista de textos.

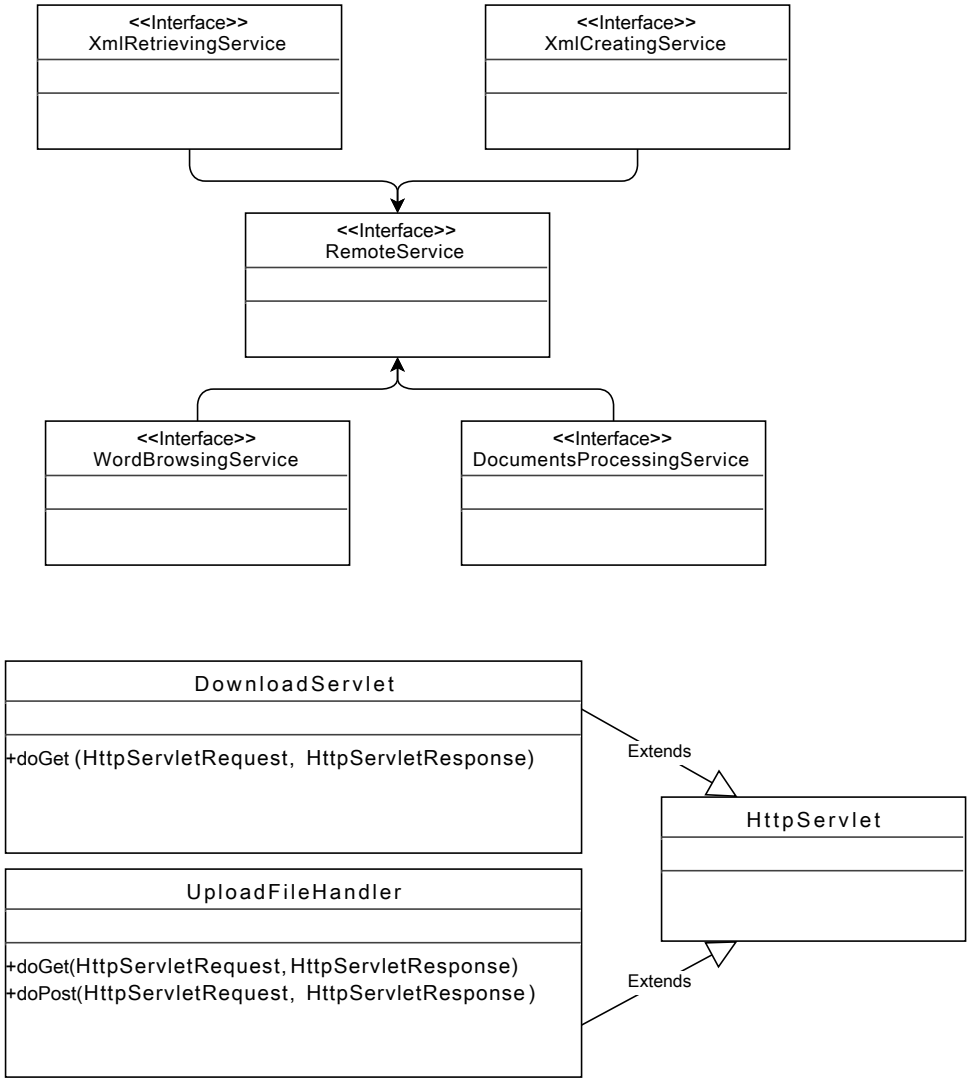


Figura IV.9: Clases relacionadas con los servlets o servicios remotos

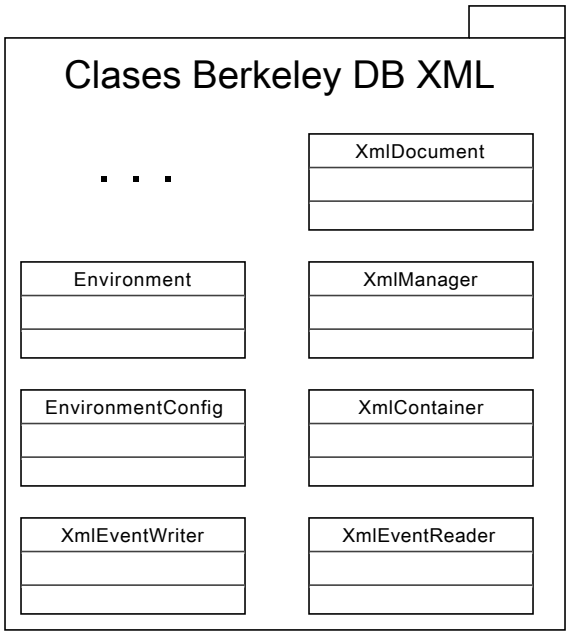


Figura IV.10: Clases necesarias para utilizar la BBDD Berkeley en nuestra interfaz

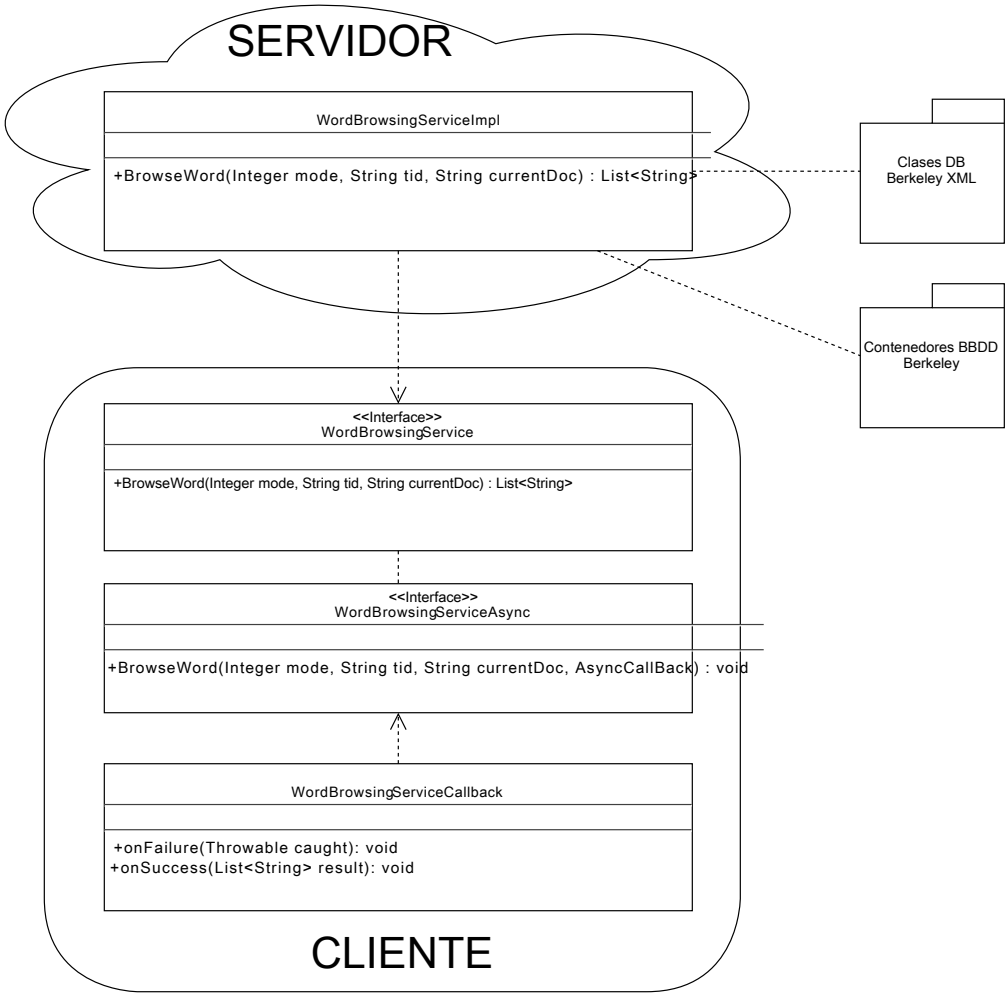


Figura IV.11: Servicio remoto para buscar términos en los documentos KAF almacenados en BBDD

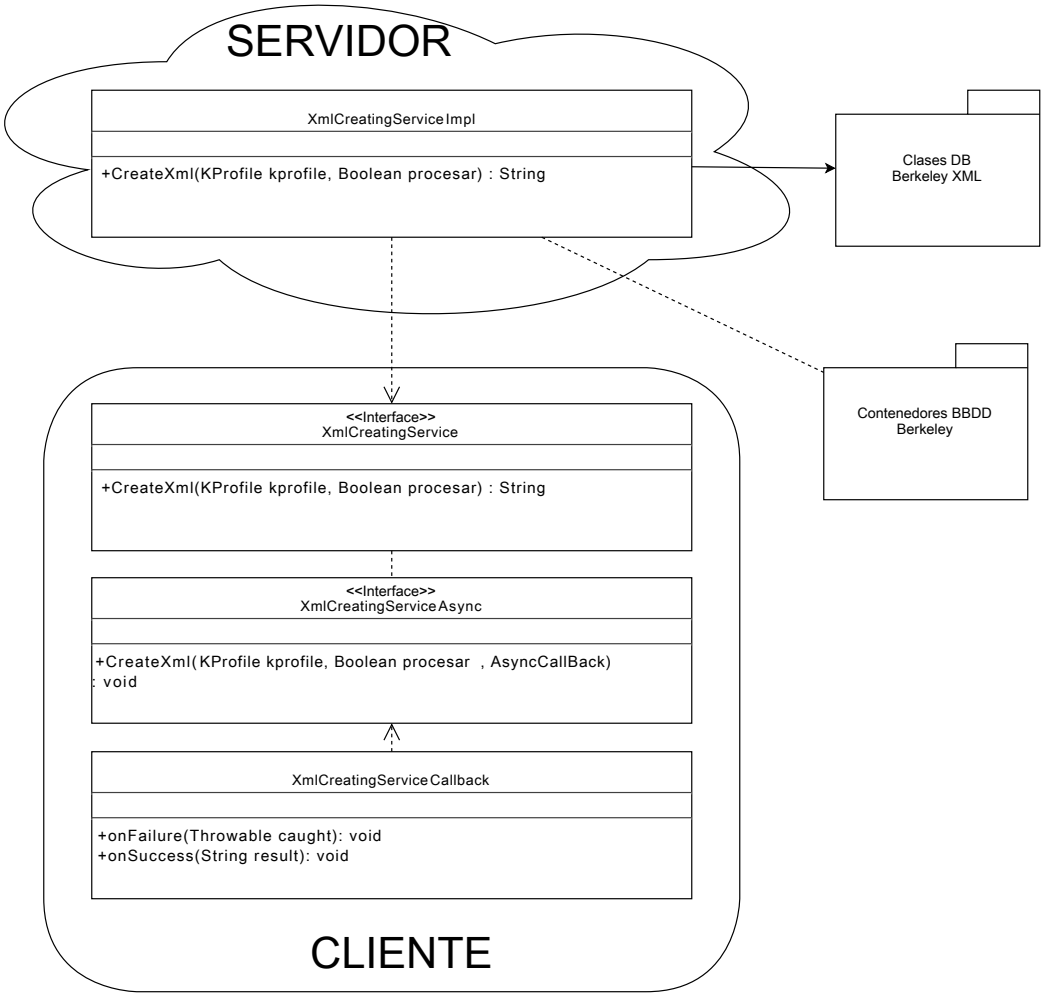


Figura IV.12: Servicio remoto para crear guardar o actualizar kybots en la BBDD

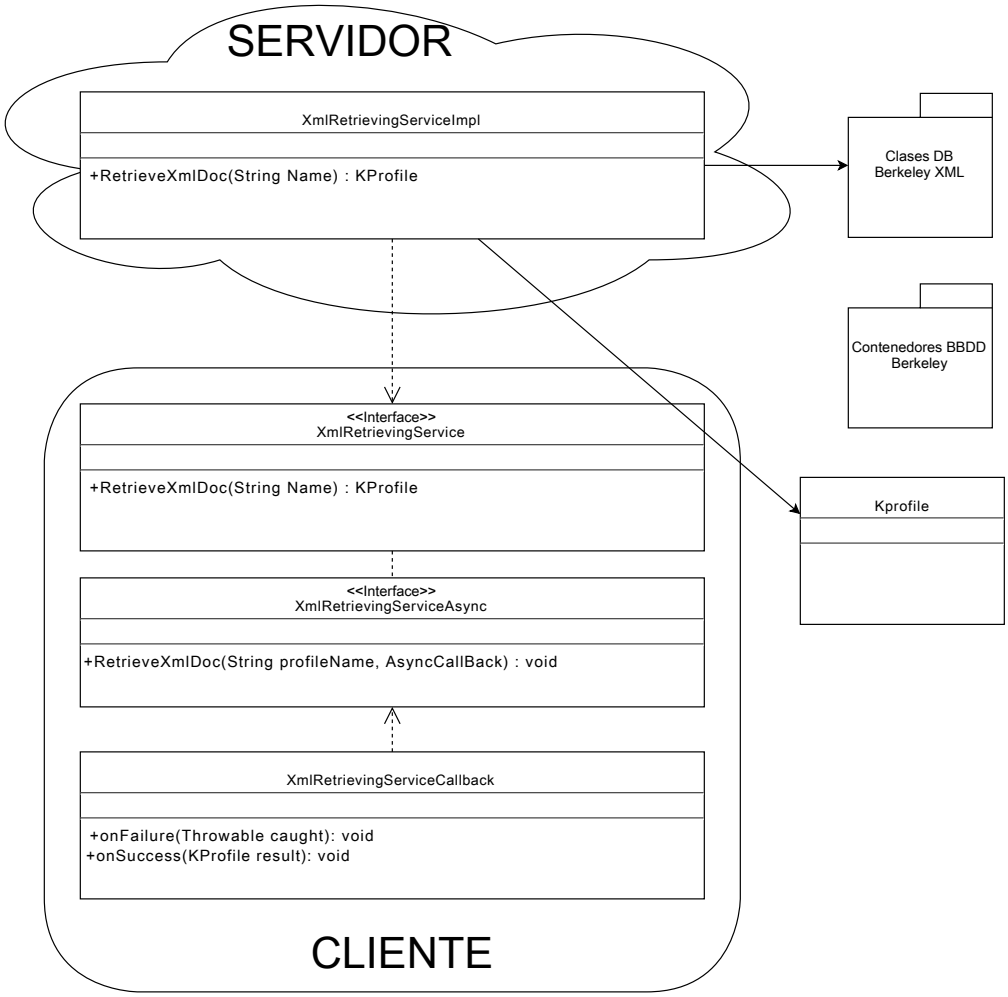


Figura IV.13: Servicio remoto de obtención de kybots de BBDD

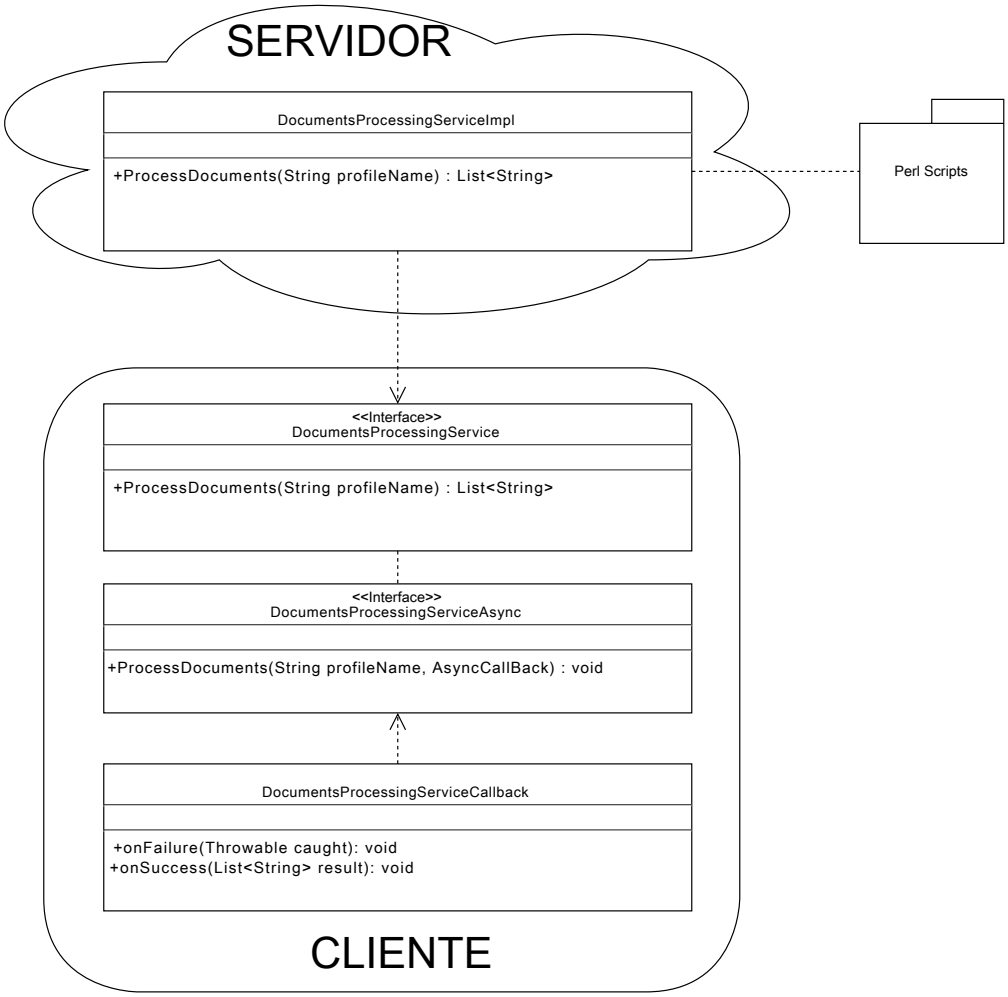


Figura IV.14: Servicio remoto para procesar un kybot

IV.3.2.2. Vistas de la interfaz

La figura IV.15 muestra las clases y elementos necesarios para poder crear las vistas de la interfaz. Además, también se muestra la clase encargada de inicializar la aplicación, y los componentes requeridos para ello. Los cuadros azules muestran las clases relacionadas con las distintas vistas. En la figura pueden verse 4, pero sólamente 3 serán visibles, ya que la clase *KybotByExample* se usa como *EntryPoint*, es decir, para inicializar la aplicación, y visualizar las otras 3 vistas. Como se explicó anteriormente, se ha separado la propia implementación de las vistas en archivos XML, como puede apreciarse. Cada uno de las 4 clases tiene su XML correspondiente donde están definidos los componentes de cada vista. La clase *UiBinder* servirá para enlazar dichos archivos con las clases Java.

IV.3.2.3. Objetos usados en la comunicación y en la visualización

La figura IV.16 muestra por una parte las clases encargadas de crear los objetos que se mostrarán en pantalla, como *TreeOfTerms*, que convierte los kybots a la estructura de árbol que se muestra en la interfaz. Por otro lado también se pueden ver las clases que se enviarán entre el cliente y servidor, como *KProfile* y *VarParameters* que componen los kybots, y *DocItem* y *ExternalRef* para los KAFs.

Finalmente, la imagen también muestra en qué vistas se usará cada servicio remoto explicado anteriormente.

IV.3.3. Diagramas de secuencia

En esta sección se mostrarán y explicarán los diagramas de secuencia por cada caso de uso definido anteriormente.

La figura IV.17 muestra la secuencia del caso de uso *Crear kybot*: Una vez el usuario ha pulsado el botón de crear un kybot, aparecerá un *popup* en el que tendrán que introducir los atributos deseados para éste. Una vez introducidos, se creará una instancia del kybot, y se realizará el proceso *MostarKybot*: Primero se limpiará el panel donde se muestran los kybots, para eliminar el que estuviese antes. Tras esto, la clase *TreeOfTerms* analizará el kybot nuevo, y creará un árbol en función a los atributos que se hayan elegido al principio en el *popup*. Una vez creado el árbol, éste se mostrará en el panel.

La figura IV.18 muestra la secuencia del caso de uso *Cargar kybot*: Una

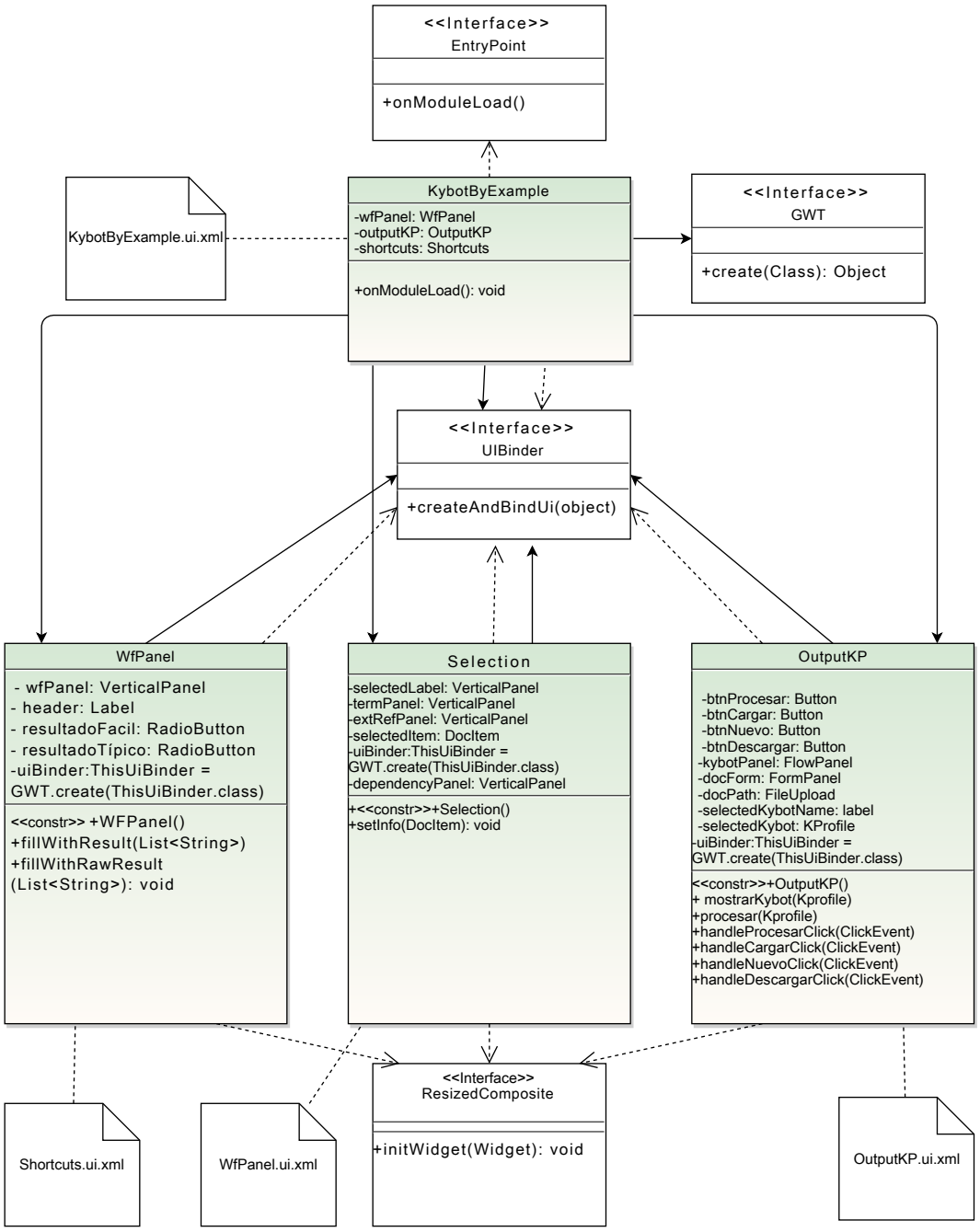


Figura IV.15: Clases relacionadas con la interfaz de la aplicación

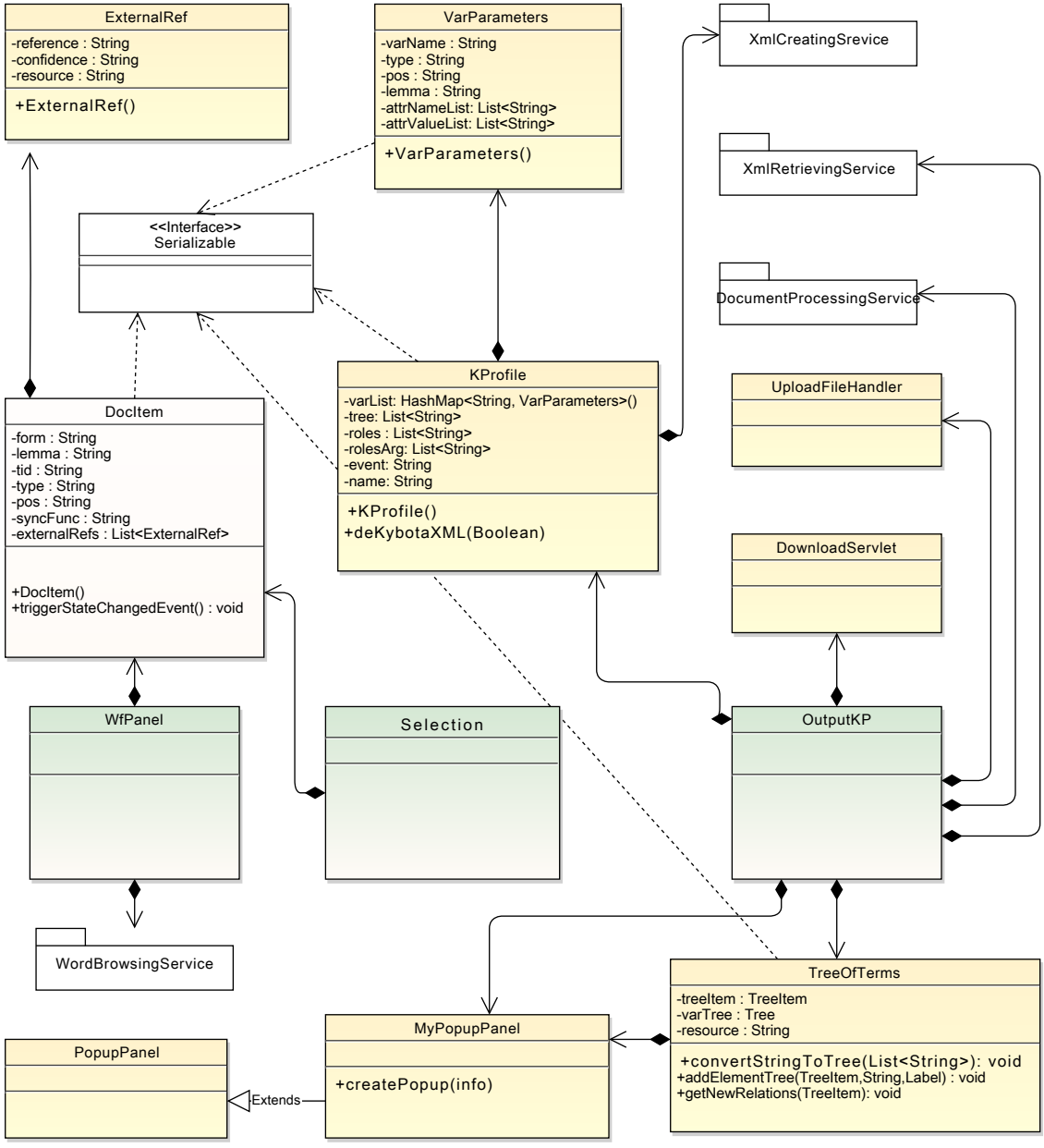


Figura IV.16: Clases de los objetos que se muestran en la interfaz, y los usados en la comunicación con el servidor

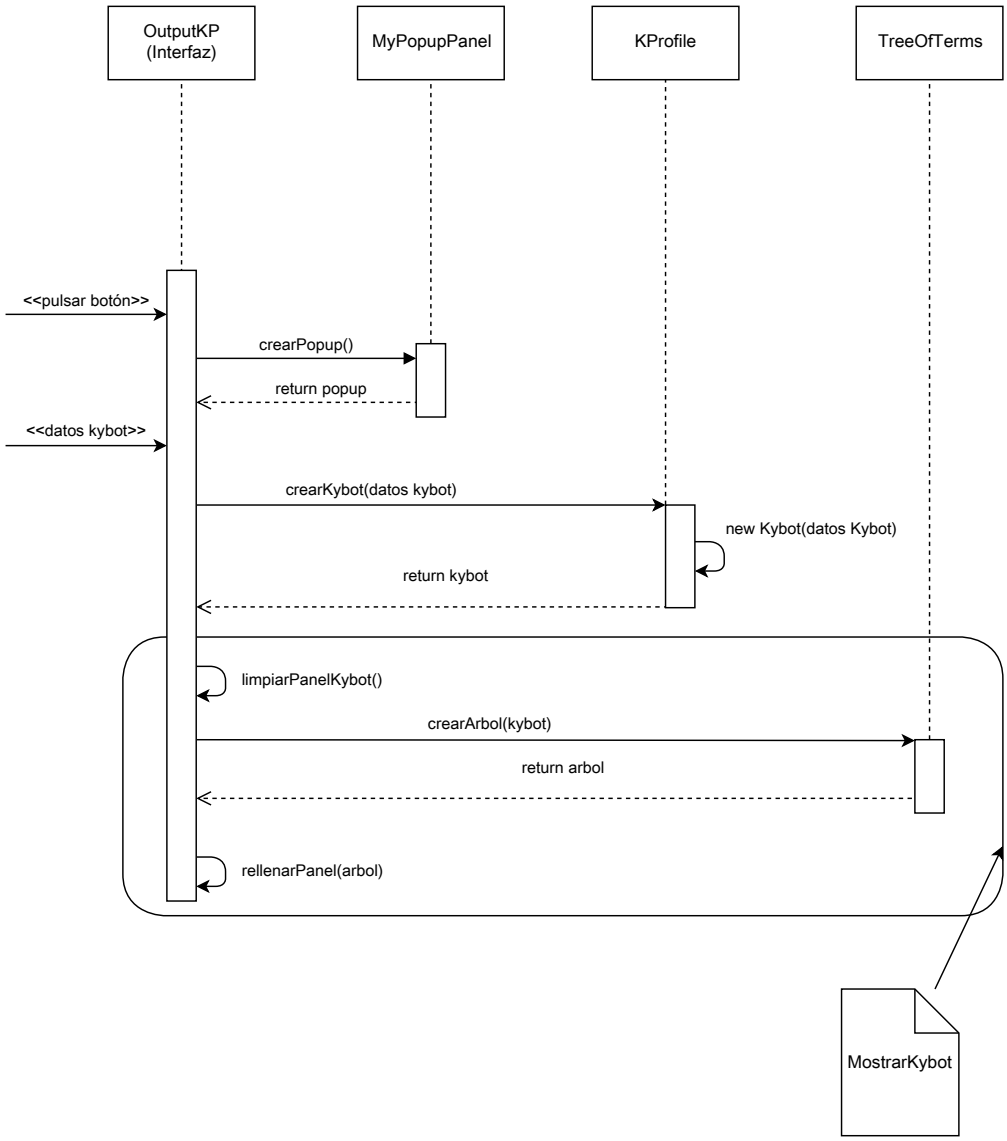


Figura IV.17: Crear kybot

vez el usuario ha pulsado la tecla de subir un kybot al sistema y se ha elegido el archivo del kybot que quiera subirse, el servlet *UploadFileHandler* se encargará de enviarlo al servidor, donde se ejecutará un script que guardará el kybot en BBDD. Una vez almacenado, se accederá a él, y se transformará del formato XML a un objeto Java. Una vez obtenido este objeto, se realizará el proceso *MostrarKybot* explicado en la figura IV.17.

La figura IV.19 muestra la secuencia del caso de uso *Ejecutar kybot* (parte 1): Una vez el usuario ha pulsado la tecla de ejecutar el kybot, éste se analizará para obtener todos sus atributos. Una vez se tienen todas sus características, se realizará el proceso *DeKybotAXML*: Se envía el kybot al servidor, donde cogiendo las características antes obtenidas, se va creando un documento XML. Éste, se añadirá/actualizará en BBDD.

La figura IV.20 muestra la secuencia del caso de uso *Ejecutar kybot* (parte 2):Una vez realizado el proceso *DeKybotAXML*, se llamará al servicio *DocumentProcessingService*, el cuál, mediante un script en Perl, ejecutará el kybot previamente guardado en BBDD sobre los documentos KAF, obteniendo así los resultados. Estos podrán visualizarse de dos formas, las cuales analizaremos a continuación.

La figura IV.21 muestra la secuencia del caso de uso *Ver output típica*: Si el usuario ha elegido visualizar los resultados de esta forma, estos aparecerán directamente en la pantalla sin ningún tipo de procesamiento y en texto plano, tal y como se mostraban en el módulo de minería. Por lo que no se podrá interaccionar con ellos.

La figura IV.22 muestra la secuencia del caso de uso *Ver output interactiva/fácil*: Si el usuario ha elegido esta opción, en vez de visualizar directamente el resultado, hará lo siguiente: Extraerá los términos e información necesaria que aparece en el resultado, y con dicha información se llamará al servicio *WordBrowsingService*, el cual buscará en BBDD dichos términos con sus atributos. Aparte de buscar dichos términos, también cogerá los 5 anteriores y posteriores de cada uno, para así entender mejor en qué contexto está cada palabra buscada. Una vez obtenidos todos los términos, se estructurarán de forma que se puedan ver correctamente en la pantalla, y se añadirán unos *listeners* para que el usuario pueda interaccionar con ellos, como veremos en la siguiente imagen. Finalmente se visualizará en pantalla el resultado. La figura VII.5 que aparece en el anexo muestra la apariencia de este tipo de *output*.

La figura IV.23 muestra la secuencia del caso de uso *Edición de kybot desde resultados obtenidos* (paso 1): Cuando se seleccione un término del re-

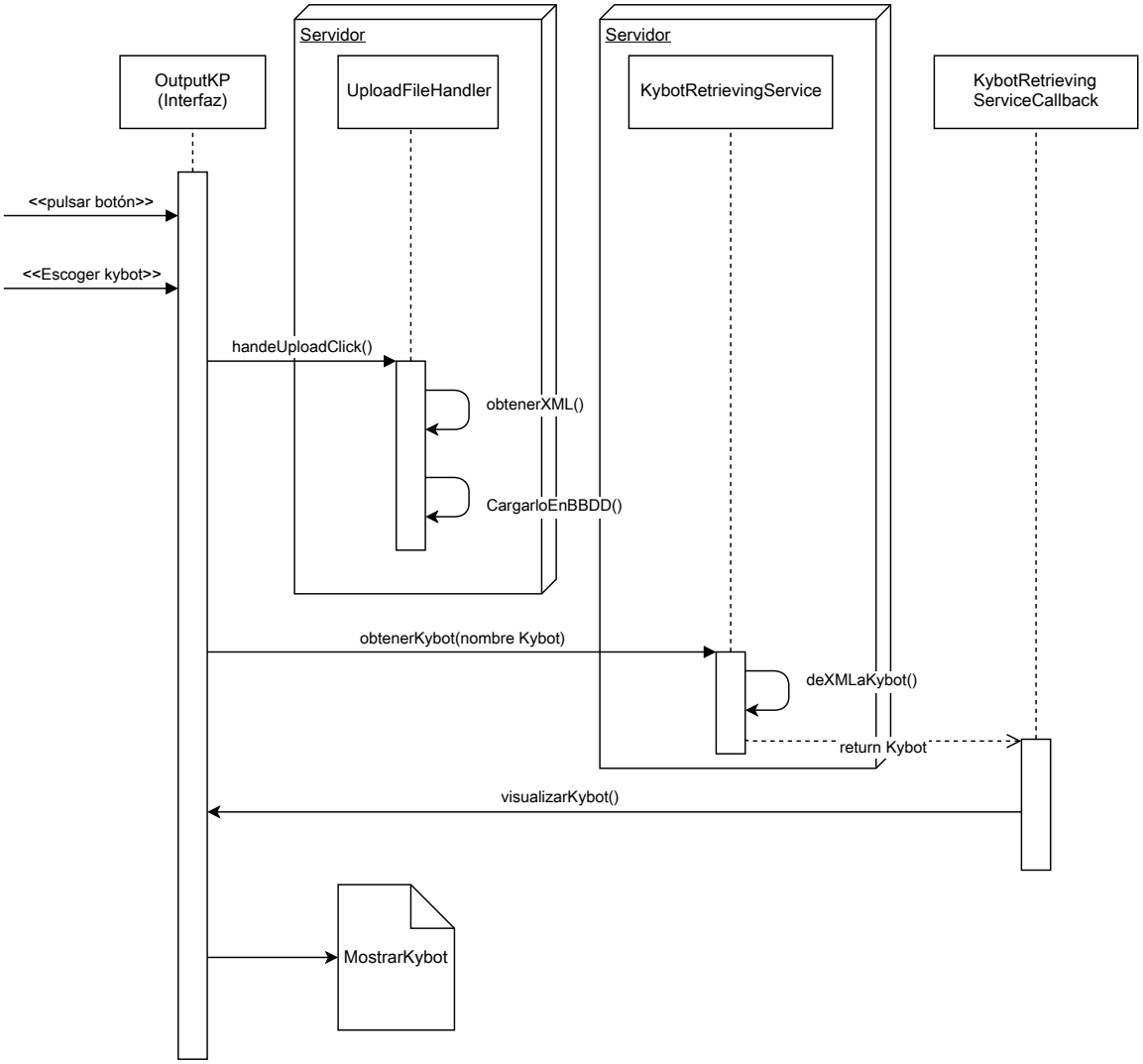


Figura IV.18: Cargar kybot

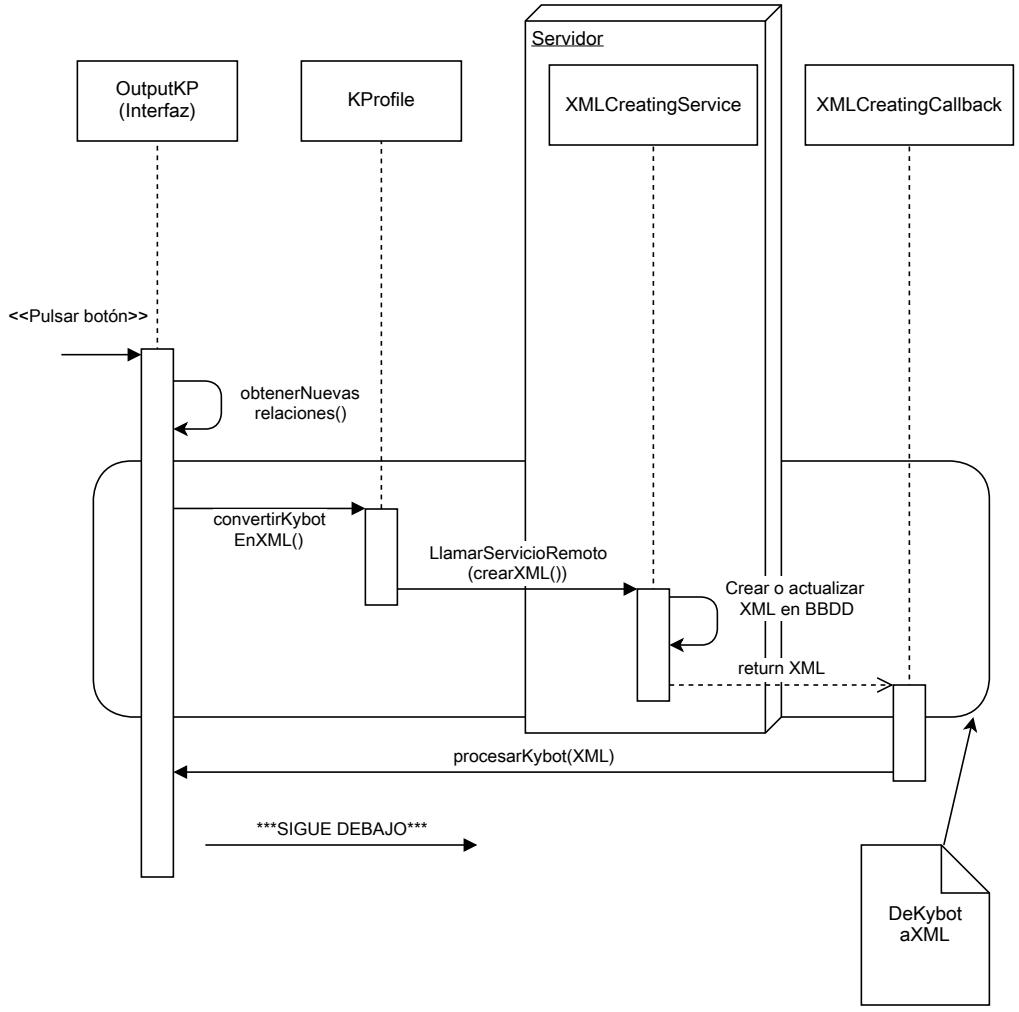


Figura IV.19: Ejecutar kybot(parte 1)

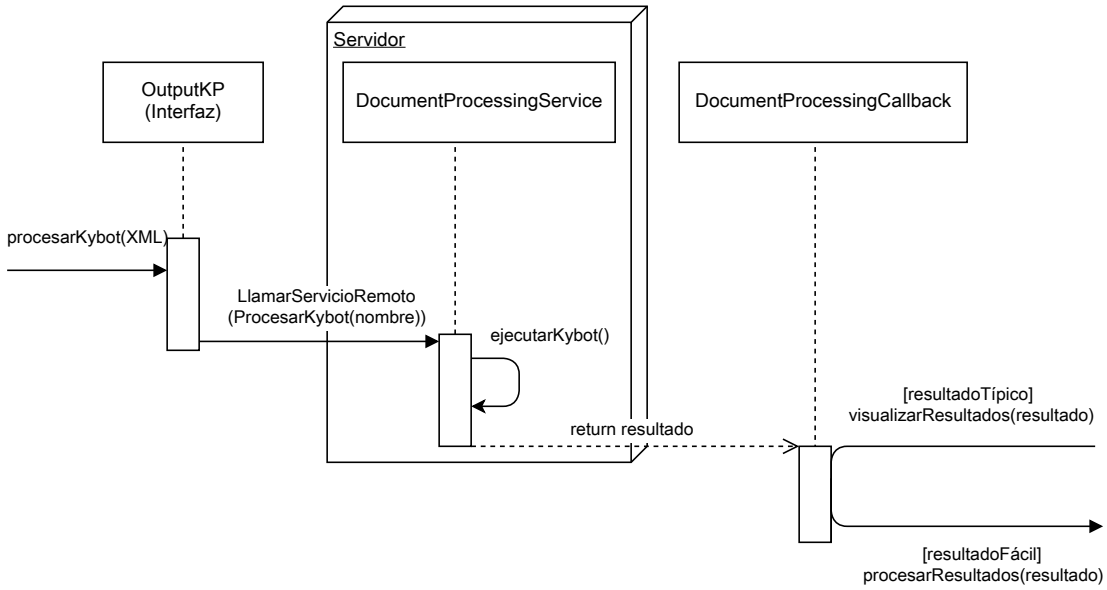


Figura IV.20: Ejecutar kybot(parte 2)

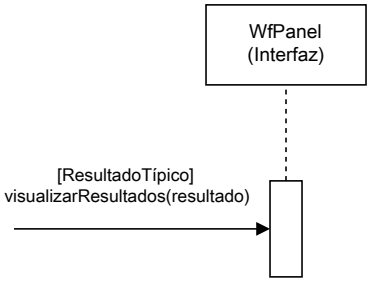


Figura IV.21: Ver *output* típica

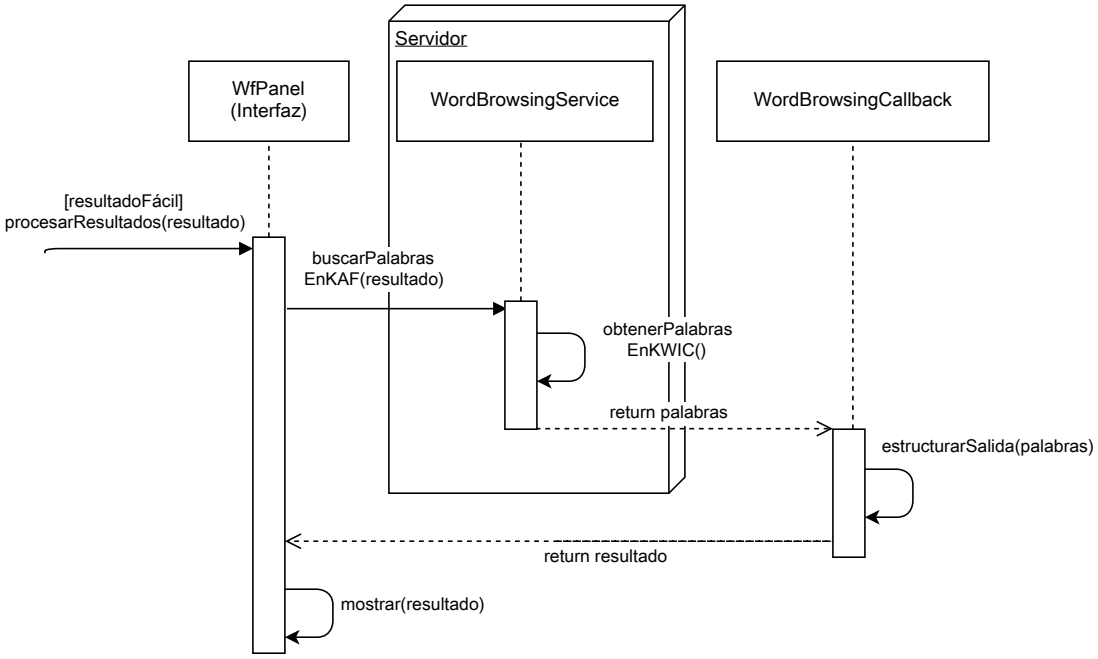


Figura IV.22: Ver *output* interactiva/fácil

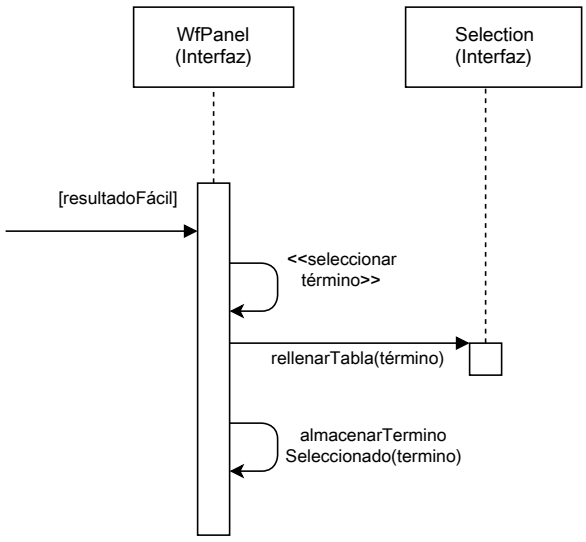


Figura IV.23: Edición de kybot desde resultados obtenidos(paso 1)

sultado obtenido, el *listener* implementado hará que una tabla de información se rellene con los datos del término seleccionado, para que el usuario pueda ver si ese término le interesa. Además, quedará almacenado en memoria por si decide añadirlo al kybot que se está mostrando en pantalla.

La figura IV.24 muestra la secuencia del caso de uso *Edición de kybot sin selección de resultados/Edición de kybot desde resultados obtenidos* (paso 2): En este punto, si pulsamos en cualquiera de los elementos del árbol del kybot, nos saldrá un *popup* con varias opciones para editarlo. Si previamente habíamos elegido algún término de los resultados interactivos, también se visualizará aquí. De lo contrario, sólo saldrán tareas tales como eliminar la variable seleccionada, añadir opciones introducidas desde el teclado, etc. Una vez escogida la opción deseada, el panel donde se muestra el kybot se actualizará usando los procesos *DeKybotAXML* y *MostrarKybot*, de esta forma el kybot se visualizará correctamente y estará preparado para volver a ser ejecutado en cualquier momento.

La figura IV.25 muestra la secuencia del caso de uso *Descargar kybot*: Una vez el usuario ha pulsado la tecla de descargar el kybot, se realizará el proceso *DeKybotaXML* explicado en la figura IV.19, se llamará al servicio *DownloadServlet*, el cual se encargará de enviar el archivo XML al servidor, y comenzará la descarga del archivo XML con el kybot.

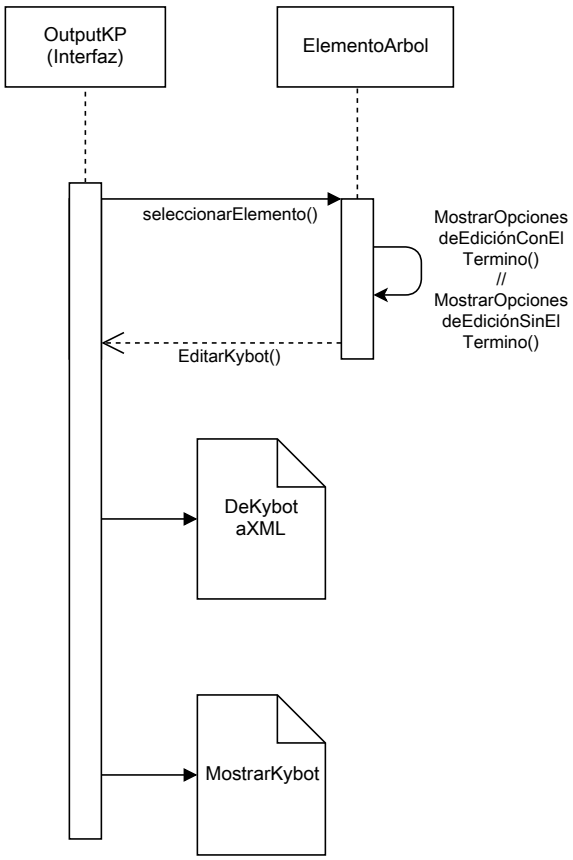


Figura IV.24: Edición de kybot sin selección de resultados/Edición de kybot desde resultados obtenidos(paso 2)

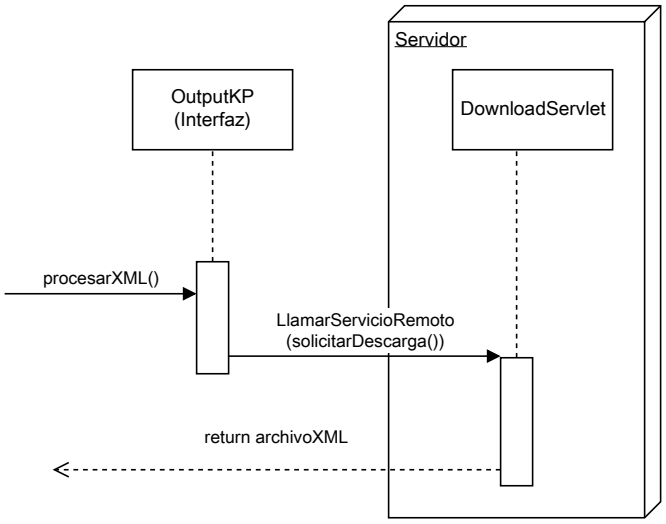


Figura IV.25: Descargar kybot

IV.4. Implementación

La aplicación se ha implementado en Linux, usando el entorno de desarrollo Eclipse, con el plugin *Google Web Toolkit* y usando el lenguaje Java para la mayor parte de la aplicación, ya que para construir la interfaz y para algunos archivos de configuración se ha usado ficheros XML. A continuación se describirá todo el proceso de implementación llevado a cabo, tanto en Eclipse, como fuera de dicho entorno.

IV.4.1. Eclipse+GWT

La figura IV.26 muestra la estructura del proyecto creado en Eclipse, que podría dividirse en los siguientes apartados:

- La carpeta *src* que contiene todo el código Java y los ficheros XML creados tanto para la configuración global de la aplicación, como para crear la interfaz de usuario. Las clases y archivos de esta sección están divididos en diferentes paquetes para una mejor gestión y organización de todo el código.

- Todas las librerías y paquetes necesarios para el funcionamiento de todos los componentes, como la BBDD Berkeley XML, el propio plugin

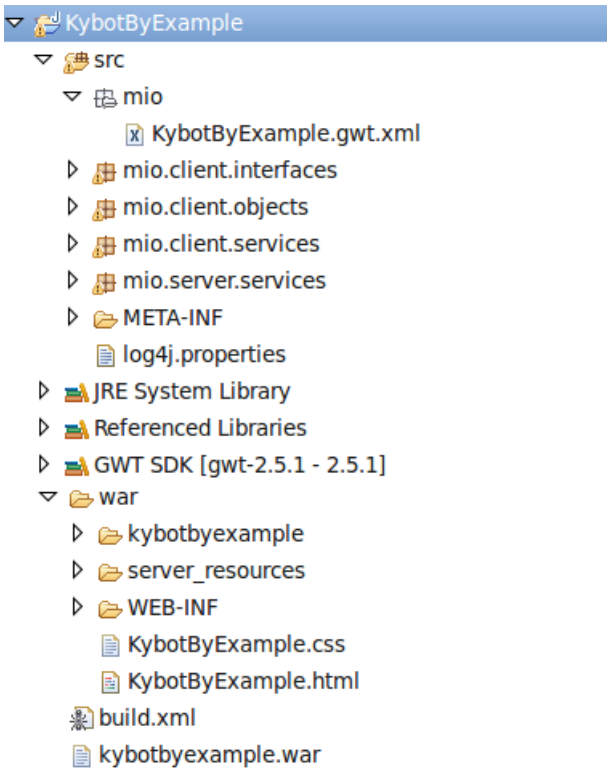


Figura IV.26: Estructura del proyecto

GWT, Java, etc.

- El archivo *Build.xml* y la carpeta *war* donde se encuentran archivos relevantes para el funcionamiento y configuración de la aplicación en el servidor.

IV.4.1.1. Carpeta src

Es la carpeta que contiene todo el código de la aplicación. Aquí también se encuentra *KybotByExample.gwt.xml*, que es el archivo de configuración de GWT en el que se especifica cuál va a ser el punto de entrada (clase que iniciará la aplicación).
Se han creado diferentes paquetes dentro de la carpeta *src* para facilitar la gestión y mantenimiento del código. A continuación explicaremos cada uno de ellos:

- *mio.client.interfaces*(figura IV.27(a)): Por una parte, se encontrarán todos los archivos requeridos para crear la interfaz de usuario:
  - Archivos *.ui.xml* en los que se definen las distintas vistas de la interfaz. La figura IV.28 muestra el archivo *.ui.xml* de la clase *OutputKP*. Como puede apreciarse, varios de los elementos definidos tienen el atributo *ui:field*. Éste será el identificador que sirva para unir los elementos con los que se definan en sus homólogos en Java.
  - Clases *.java* que complementen a sus parejas *.ui.xml*. En estas clases se definirá el comportamiento de los elementos entre otras cosas. Como hemos dicho antes, para lograr unir los elementos de ambos archivos, tendremos que usar el atributo *ui:field* definido anteriormente. La figura IV.29 nos muestra cómo unir los elementos de *outputKP.ui.xml* con los de *outputKP.java*. Para ello, será necesario extender la clase *UiBinder*, y crear una instancia. Una vez relizado, el enlace se realizará mediante anotaciones como *@UiField* para los elementos y *@UiHandler* para el tema de control. En la figura, se puede ver cómo se define *handleLoadClick()*, cuando pulsemos sobre el botón *btnCargar*, también definido en el archivo xml e identificado con *ui:field*.
  - Todos los elementos multimedia usados. En nuestro caso unas cuantas imágenes para usarlas como iconos.

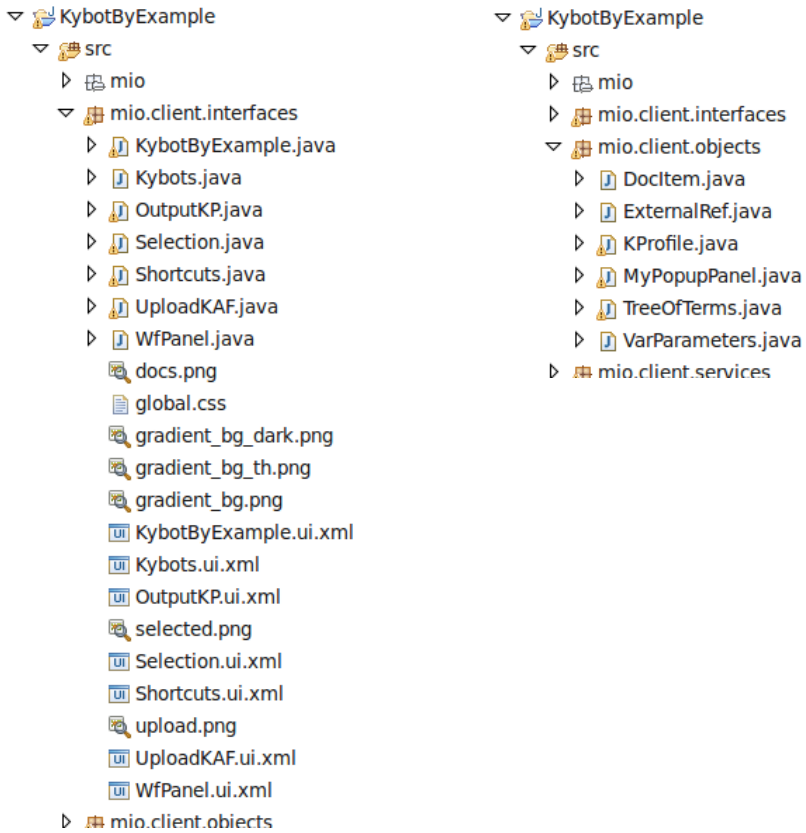


Figura IV.27: Paquete de la interfaz (izquierda) y paquete de los objetos (derecha)

```
<g:DockPanel width="100%" height="30px" styleName='{style.header}'
    verticalAlignment="ALIGN_MIDDLE">
    <g:Dock direction="WEST">
        <g:HorizontalPanel horizontalAlignment="ALIGN_CENTER"
            verticalAlignment="ALIGN_TOP" spacing="15">
            <g:Label styleName='{style.gris}' ui:field='lblPerfil'
                text="Perfil Kybot:" />
            <g:Label text=" " ui:field='header' width="100%" />
            <g:Button ui:field='btnGuardar' height="31px">Guardar
            </g:Button>
            <g:Button ui:field="btnNuevo">Nuevo</g:Button>
            <g:Button ui:field='btnProcesar' height="31px">Procesar
            </g:Button>
            <g:FormPanel ui:field="docForm">
                <g:FlowPanel ui:field="inputPane">
                  <g:FileUpload ui:field="docPath" />
                  <g:Button ui:field='btnCargar' height="31px">Cargar
                  </g:Button>
                </g:FlowPanel>
            </g:FormPanel>
        </g:HorizontalPanel>
    </g:Dock>
</g:DockPanel>
```

Figura IV.28: Fragmento de OutputKP.ui.xml

- Por último, un archivo *.css* donde se definan algunos estilos globales para la interfaz, los cuales se cargarán en el método *onModuleLoad()*, que explicaremos a continuación.

La apariencia final de las vistas de la interfaz puede apreciarse en el anexo del manual de usuario, concretamente en la sección VII.1.3.

Por otra parte, además de todos los elementos necesarios para construir la interfaz, este paquete también contiene la clase *KybotByExample.java* (figura IV.30), que implementa la interfaz *EntryPoint* (punto de entrada) y el método *onModuleLoad()*. Este método será el que arranque la aplicación. Como se ha mencionado antes, se tiene que especificar en el archivo *KybotByExample.gwt.xml* cuál será esta clase.

```
interface Binder extends UiBinder<Widget, OutputKP> {}

private static final Binder binder = GWT.create(Binder.class);

@UiField
Label lblPerfil;

@UiField
Button btnCargar;

@UiField
public static FlowPanel kybotPanel;

static List<String> resultado = new ArrayList<String>();

@UiHandler("btnCargar")
void handleLoadClick(ClickEvent e) {
```

Figura IV.29: Fragmento de OutputKP.java

```
public class KybotByExample implements EntryPoint {
    interface Binder extends UiBinder<DockLayoutPanel, KybotByExample> {}
    interface GlobalResources extends ClientBundle {
        @Source("global.css")
        CssResource css();
    }
    private static final Binder binder = GWT.create(Binder.class);

    @UiField
    WfPanel wfPanel;
    @UiField
    OutputKP outputKP;
    @UiField
    Shortcuts shortcuts;

    public void onModuleLoad() {

        GWT.<GlobalResources> create(GlobalResources.class).css()
                .ensureInjected();
        DockLayoutPanel outer = binder.createAndBindUi(this);
        Window.enableScrolling(false);
        Window.setMargin("0px");
        outputKP.setListener(new OutputKP.Listener() {
            @Override
            public void onVarClicked(VarParameters item, int x, int y) {
                MyPopupPanel.MyPopup panel = new MyPopupPanel.MyPopup(item);
                panel.setPopupPosition(x, y);
                panel.show();
            }
        });
        com.google.gwt.core.client.GWT.isClient();
        RootLayoutPanel root = RootLayoutPanel.get();
        root.add(outer);
    }
}
```

Figura IV.30: Módulo de entrada

Figura IV.31: Representación del kybot en la interfaz, usando la clase *TreeOfTerms.java*

- *mio.client.objects*(figura IV.27(b)): Aquí están las clases usadas para transportar la información entre el cliente y servidor, las cuales tienen que implementar la interfaz *Serializable* para que puedan enviarse a través del protocolo de GWT. Como puede apreciarse en la figura IV.27(b), aquí tendríamos las clases que representan a los kybots (*KProfile.java* y *VarParameters.java*) y elementos de los documentos KAF (*DocItem.java* y *ExternalRef.java*). La figura IV.16 muestra con más detalle los atributos y las relaciones de estas clases.

  Por otra parte este paquete también alberga las clases auxiliares para mostrar elementos en pantalla, como *TreeOfTerms.java*, que sirve para representar el kybot en pantalla de una forma más intuitiva. La figura IV.31 muestra un ejemplo de cómo se representan los kybots usando la estructura de árbol. Por otra parte, en la sección 1.3 del anexo se explica como son representados los kybots usando esta estructura.

- *mio.client.services*(figura IV.32(a)): Aquí irán las clases necesarias en la parte del cliente para crear los servicios remotos de GWT. Para esto, se deberán definir cuatro componentes java, tres de ellos en la parte del cliente, y la cuarta en el servidor. A continuación se explicará cuál es la funcionalidad de cada una de estas tres clases, y se mostrará un ejemplo práctico de cómo invocar el servicio remoto.

  - Lo primero será crear una interfaz que extienda a *RemoteService*, para que el compilador entienda que se trata de un servicio remoto de GWT. Aquí se definirán los diferentes métodos de comunicación. Por ejemplo, dado un nombre de kybot desde el cliente, que

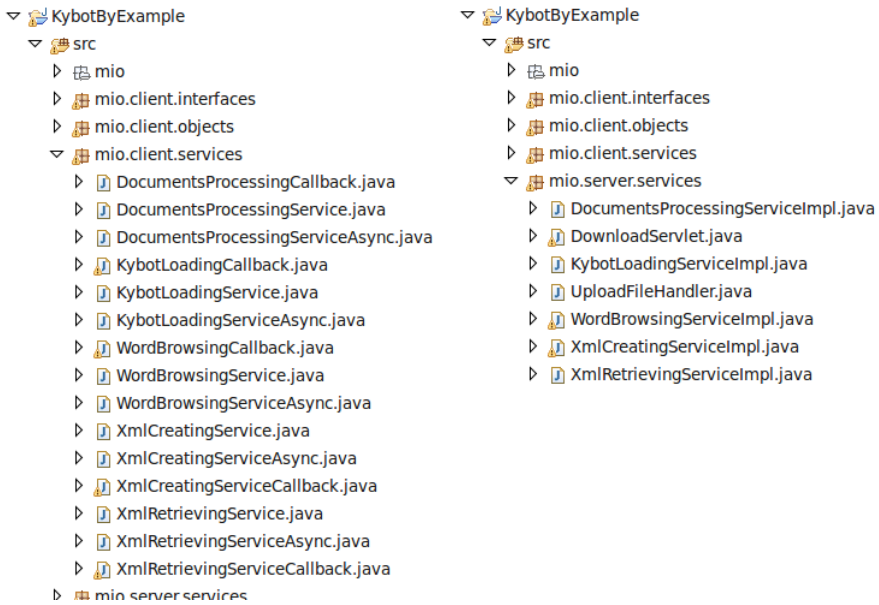


Figura IV.32: Paquete de servicios en la parte del cliente (izquierda) y paquete de servicios en la parte del servidor (derecha)

el servidor busque en BBDD dicho kybot y lo devuelva en el objeto correspondiente. La implementación de esta interfaz, es decir, el código de cada una de las operaciones definidas en esta clase, irá en el servidor como veremos más adelante.

- Versión asíncrona de la interfaz recién creada. El compilador creará automáticamente esta clase y será la usada una vez se invoque cada servicio, ya que estós serán asíncronos.
- Clase que se encargará de recibir la resputesta del servidor. Deberán implementarse dos métodos obligatoriamente: *OnFailure()*, que se ejecutará cuando haya algún problema durante la llamada, y *OnSuccess()*, que devolverá los resultados deseados en caso de que el servicio remoto transcurra correctamente.

- *mio.server.services*(figura IV.32(b)): En este paquete irá última clase de las cuatro que forman el servicio remoto. Ésta será la que implementará los métodos que hemos definido en la interfaz y deberá extender la clase *RemoteServiceServlet*. Ésta, servirá para deserializar los objetos

que reciba desde el cliente y serializarlos al enviarlos de vuelta. Por otra parte también tenemos aquí los servlets de carga y descarga de kybots, en los que sólo hace falta una clase en la parte del servidor para su funcionamiento.

Por tanto, el código desde el que se accederá a la BBDD Berkeley o a los scripts del módulo de minería irá aquí, en el servidor, en las clases que implementan los servicios remotos.

Para realizar cualquier consulta en la BBDD Berkeley, primero habrá que inicializar y configurar el entorno de la BBDD. Para ello, se creará una instancia de la clase *EnvironmentConfig()*, y ahí configuraremos parámetros como la gestión de *logs*, soporte transaccional, etc.

Tras esto, crearemos una instancia de la clase *XMLManager*, la cual servirá para gestionar los *containers* (estructuras donde están guardados los kybots y KAFs) y sus documentos. También nos servirá para preparar y ejecutar llamadas *XQuery*, por lo que es un paso muy importante. Una vez se haya creado y configurado correctamente lo anterior, ya podrá realizarse cualqueir tipo de consulta *XQuery* en la base de datos. Todo el proceso recién explicado puede apreciarse en la figura IV.33(b).

Por otra parte, también pueden ejecutarse directamente archivos script, como se aprecia en la figura IV.34. Para ello, se tendrá que crear una instancia de las clases *Runtime* y *Process*. Ésta última ejecutará el script que le pasemos mediante una cadena de texto, y mediante la clase *BufferedReader* iremos obteniendo los resultados de la ejecución. Los scripts que se usarán a lo largo del proceso se pueden encontrar en la carpeta *war/server_resources*.

Una vez se tengan las cuatro clases implementadas, debería indicarse en el archivo *WEB-INF/web.xml*, que veremos más adelante, la ruta de cada servicio remoto. Una vez se haya hecho, ya se podrán hacer llamadas asíncronas desde el cliente, del mismo modo que aparece en la figura IV.35. Estos son los pasos más importantes para realizar la llamada:

1. Crear una instancia del servicio que queramos usando *GWT.create (clase deseada)*

2. Inicialiación del *callback*, es decir, de la clase que obtendrá la respuesta. En este caso se ha definido previamente la variable resultado, que

```
EnvironmentConfig envConf = new EnvironmentConfig();
envConf.setAllowCreate(false);
envConf.setInitializeCache(true);
envConf.setInitializeLocking(true); // Turn on the locking
                                    // subsystem.
envConf.setInitializeLogging(true); // Turn on the logging
                                    // subsystem.
envConf.setTransactional(true);
XmlRetrievingServiceImpl.myEnv = new Environment(
        XmlRetrievingServiceImpl.envHome, envConf);
XmlManagerConfig managerConfig = new XmlManagerConfig();
managerConfig.setAllowAutoOpen(true);
managerConfig.setAllowExternalAccess(true);
XmlRetrievingServiceImpl.myManager = new XmlManager(
        XmlRetrievingServiceImpl.myEnv, managerConfig);
// Open a container
XmlRetrievingServiceImpl.myContainer = XmlRetrievingServiceImpl.myManager
        .openContainer("/home/dani/mining_module/dbxml/docs_en.dbxml");

   XmlQueryContext context = XmlRetrievingServiceImpl.myManager
           .createQueryContext();

   List<String> sentString = new ArrayList<String>();
   if (mode == 2) {
       String[] tidParts = tid.split("_");
       Integer tidNumber = Integer.parseInt(tidParts[0].substring(1));

       String myQuery = "for $i in doc('dbxml:/docs_en.dbxml/"
               + currentDoc + "')/KAF2/*/*/*/*\n";
       myQuery += "where $i/@tid='t" + tidNumber + "_" + (tidParts[1])
               + "_es_'\n";
       myQuery += "order by $i/@tid\n";
       myQuery += "return $i";

       XmlQueryExpression qe = XmlRetrievingServiceImpl.myManager
               .prepare(myQuery, context);
       XmlResults results = qe.execute(context);

       while (results.hasNext()) {
           XmlValue va = results.next();
           RESULT.add(va.asString());
       }
```

Figura IV.33: Código necesario para inicializar la BBDD Berkeley y realizar consultas

```
Runtime rt = Runtime.getRuntime();
Process proc = rt
        .exec("/home/dani/workspace/KybotByExample/war/server_resources/kybot_run.pl "
        + " --container-name docs_en --kybot-container-name kybots_en " + profileName);

BufferedReader stdInput = new BufferedReader(new InputStreamReader(
        proc.getInputStream()));

BufferedReader stdError = new BufferedReader(new InputStreamReader(
        proc.getErrorStream()));

// read the output from the command
System.out.println("Here is the standard output of the command:\n");
while ((s = stdInput.readLine()) != null) {
    System.out.println(s);
    result.add(s);
}
```

Figura IV.34: Pasos necesarios para ejecutar los scripts del módulo de minería en el servidor

```
DocumentsProcessingServiceAsync service = (DocumentsProcessingServiceAsync) GWT
        .create(DocumentsProcessingService.class);
ServiceDefTarget serviceDef = (ServiceDefTarget) service;
serviceDef.setServiceEntryPoint(GWT.getModuleBaseURL()
        + "documentsProcessingService");
AsyncCallback<List<String>> myUserCallback = new DocumentsProcessingCallback(
        resultado);
service.ProcessDocuments(header.getText(), myUserCallback);
```

Figura IV.35: Protocolo necesario para realizar una RPC

será donde se almacene lo que nos devuelva el servidor, en caso de que todo funcione correctamente.

3. Finalmente, llamar al servicio asíncrono junto con los parámetros requeridos para que empiece la operación.

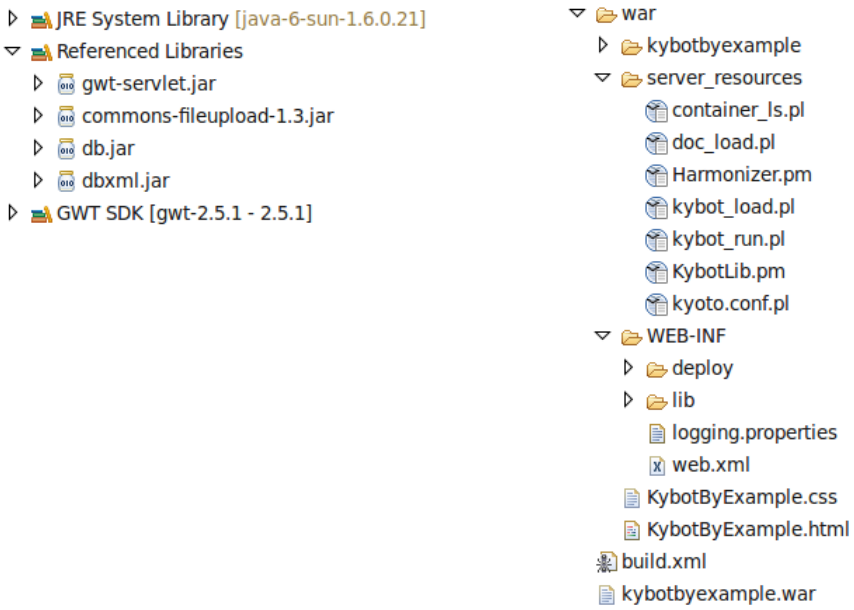


Figura IV.36: Librerías usadas en el proyecto (izquierda) y carpeta *war* del proyecto

IV.4.1.2. Librerías y paquetes

La figura (figura IV.36(a)) muestra las librerías necesarias para hacer funcionar los diferentes componentes de la aplicación. Aparte de las básicas, por un lado tenemos las relacionadas con GWT, las cuales servirán para la creación de la interfaz y la comunicacíon con el servidor. Por otro lado, *db.jar* y *dbxml.jar* son las necesarias para hacer funcionar la BBDD Berkeley en el servidor.

IV.4.1.3. Carpeta war

Esta carpeta tiene varios tipos de ficheros como puede apreciarse en la figura (figura IV.36(b)). Explicaremos los más relevantes:

- La carpeta *server_resources* contiene varios de los scripts en Perl del módulo de minería. Entre ellos están los que se ejecutarán para relizar tareas con los kybots, y los de configuración o auxiliares, como *kyoto.conf.pl*, que contendrá la ubicación del resto de archivos necesarios

para el funcionamiento de módulo de minería y otros parámetros importantes. El resto de ficheros del módulo de minería están ubicados fuera del directorio del proyecto, como se verá más adelante.

- El archivo *build.xml*, contiene el script con el que se empaquetará todo para crear el archivo *kybotbyexample.war* y alojarlo en un contenedor Tomcat. Básicamente, el script hará lo siguiente:

  1. Compilar el código java usando javac.
  2. Invocar el compilador GWT para que convierta nuestro código a Javascript.
  3. Empaquetar el código compilado en un archivo *.JAR*.
  4. Preparar y almacenar los recursos estáticos del servidor en un directorio determinado.
  5. Empaquetar todo lo anterior más las librerías necesarias en un archivo *.WAR*.

- La carpeta *deploy*, almacena los archivos *.class* creados en el 4. paso del script recién explicado.

- La carpeta *lib*, contiene las librerías que se tendrán que usar desde la parte del servidor.

- El archivo *web.xml*, dentro de la carpeta *WEB-INF* contiene la información sobre la ruta de cada servlet o servicio remoto que se usará en la aplicación. En nuestro caso, todos estos archivos estarán en la ruta *mio/server/services* gracias a la gestión de paquetes que hemos explicado anteriormente. Además, el archivo *web.xml* también define la ruta del archivo *.html* que se cargará nada más comenzar la aplicación.

- El archivo *KybotByExample.html* es la página que se mostrará cuando se arranque la aplicación. Cabe destacar que este *.html* está prácticamente vacío, ya que todo lo relacionado con la interfaz y la estructura de la página se realiza en los archivos *.ui.xml* situados en el paquete de interfaz mencionado anteriormente.

IV.4.2. Apache Tomcat + Archivos restantes de Berkeley y módulo de minería

Por otra parte, fuera del proyecto creado en eclipse, habrá dos componentes más que serán necesarios para el funcionamiento de la aplicación:

- Contenedor de servlets Apache Tomcat.
- Archivos relacionados con el módulo de minería y BBDD Berkeley XML que no han sido introducidos dentro del proyecto.

IV.4.2.1. Apache Tomcat

Para desplegar correctamente el archivo *.war* mencionado anteriormente, será necesario tener correctamente instalado y configurado Apache Tomcat en el equipo. Para ello, se realizaron los siguientes pasos:

1. Escoger un directorio para su instalación.
2. Configurar las variables *CATALINA_HOME* y *PATH*, indicando el directorio raíz donde se tenía instalado Tomcat.
3. Verificar la correcta instalación invocando el comando *catalina.sh*, el cual inicia el servidor Tomcat.
4. Una vez invocado el comando, abrir una ventana del navegador web y visitar la URL *http://localhost:8080* para comprobar que sale la página de documentación de Tomcat.

Una vez realizados estos pasos, el servidor Tomcat estará instalado correctamente y podrá ejecutarse el script *build.xml* explicado anteriormente.

IV.4.2.2. Archivos del módulo de minería

Tal y como se ha visto, parte de los ficheros del módulo de minería, entre ellos los scripts que se ejecutarán en la aplicación, se encuentran dentro del proyecto creado en Eclipse. El resto se ha almacenado fuera del area de trabajo de Eclipse, en un directorio llamado *mining_module*, cuya estructura puede verse en la figura IV.37. Aquí se encontrarán los siguientes contenidos:

- Contenedores de Berkeley con los KAF y kybots almacenados.

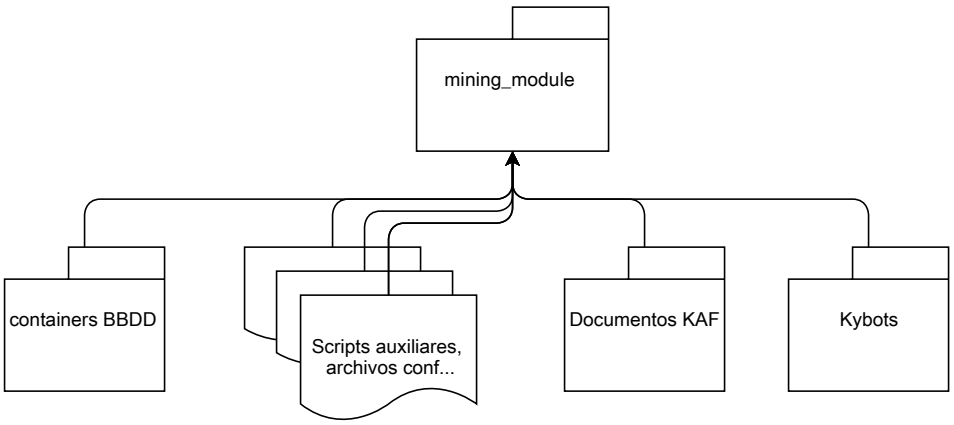


Figura IV.37: Resto de archivos necesarios para el funcionamiento del módulo de minería

- Scripts y ficheros auxiliares para el funcionamiento de la BBDD y el módulo.
- Documentos KAF que quieran cargarse manualmente (sin usar la interfaz) en la aplicación.
- Kybots que quieran cargarse manualmente en el sistema.

Para que los scripts alojados en el directorio del proyecto sepan dicha ubicación, el archivo *kyoto.conf.pl* situado en la carpeta *war/server_resources* indicará dónde se encuentran.

IV.5. Pruebas

A continuación se presentarán las pruebas que fueron realizadas para asegurar el correcto funcionamiento de los componentes del sistema. En caso de haber tenido fallos o problemas graves con alguno de los componentes a la hora de realizar pruebas, se explicarán con más profundidad en la sección V.2, donde aparecerán las incidencias más importantes ocurridas durante el desarrollo de la aplicación.

IV.5.1. Operaciones

Por cada operación que se añadía a la interfaz, se realizaba una prueba aislada de dicho componente para ver si funcionaba correctamente. En caso afirmativo, ésta se integraba junto con las otras operaciones en la aplicación. Este proceso se repitió para todas las operaciones que aparecen en los casos de uso definidos.

Cabe aclarar que antes de nada, lo primero fue probar el correcto funcionamiento de los plugins de GWT y GAE en Eclipse, que es la base donde se contruyó la aplicación. Para ello se realizó una aplicación web básica[1] y se alojó en internet usando el plugin de GAE para comprobar que todos los componentes funcionaban correctamente. Una vez logrado esto, se empezaron a implementar y a probar las funcionalidades por separado mencionadas arriba.

IV.5.2. Compatibilidad Linux y sus componentes

Al realizarse la migración de Windows a Linux, se tuvo que probar la compatibilidad de cada herramienta y componente usado en Windows. La mayor parte de los componentes no dieron problemas, pero a la hora de intentar instalar la BBDD Berkeley hubo muchos debido a incompatibilidades con ciertas librerías y ficheros. En la sección V.2 se explica más detalladamente.

IV.5.3. Integración del módulo de minería en aplicación

Una vez instalado el módulo de minería y la BBDD Berkeley XML, tuvo que comprobarse si funcionaban correctamente junto con la aplicación

[1] http://lector-kaf.appspot.com/

en GWT. Para ello se intentaron ejecutar los scripts en Perl integrados en el módulo para ver si funcionaba correctamente la BBDD Berkeley XML. Primero, se descubrió que dichos scripts del módulo sólo podían ejecutarse en la parte del servidor, y más tarde se comprobó que GAE ponía muchas dificultades para ejecutarlos también allí. Se explica más detalladamente en la sección V.2.

IV.5.4. Instalación y despliegue del war con Apache Tomcat

Por el problema surgido en el anterior punto, se cambió el componente de alojamiento de servlets de GAE a Tomcat, ya que éste sí que permitía la ejecución de scripts en el servidor. De todas formas, una vez instalado, se comprobó si había algún otro tipo de problema similar. Además, se hicieron pruebas para ver si el script *Build.xml*, descrito en el capítulo de implementación, desplegaba corectamente el archivo *.war*. En este caso no hubo problemas de ningún tipo.

IV.5.5. Usabilidad interfaz

Este tipo de pruebas se hicieron para obtener una mejor experiencia con la interfaz, más que para encontrar fallos. El objetivo de las pruebas era experimentar diferentes configuraciones de página para ver cuál era más usable e intuitiva y así lograr una mejor experiencia, que a fin de cuentas era uno de los objetivos marcados inicialmente.

IV.6. Implantación

La aplicación, como ha podido verse en la sección de implementación, a pesar de estar preparada para ser alojada en internet, por el momento sigue instalada en el ordenador del alumno, por lo que sólo puede accederse a ella mediante una red de área local. De todas formas, en un futuro cercano se espera implantarlo en un entorno donde los usuarios de KYOTO tengan acceso a ella. Para ello, se tendrá que realizar lo siguiente una vez sea implantada:

- Una vez instalado en el nuevo entorno, antes de ponerlo a disposición de los usuarios, habrá que realizar las pruebas de integración e implantación necesarias para verificar el correcto funcionamiento de la aplicación.

- Distribuir entre los usuarios el manual que aparece en el anexo de este documento.
- Llevar a cabo un plan para saber saber qué hacer en el momento que se vayan añadiendo mejoras/funcionalidades a la aplicación (cómo actualizar el entorno, informar a los usuarios, etc.).

V. CAPÍTULO

Seguimiento

Aquí se analizará si el proyecto ha transcurrido de acuerdo con lo planificado inicialmente. Como se verá, debido a varias incidencias y otros factores, se ha tenido que invertir más tiempo de lo planificado, alargando así la duración del proyecto alrededor de dos meses. A continuación analizaremos en más profundidad las razones y las consecuencias de todo esto.

V.1. Comparación con la planificación inicial

Analizando el diagrama de Gantt planificado inicialmente (figura V.1), y comparándolo con el realizado al final del proyecto (figura V.2), que refleja las horas que realmente se han invertido, lo más destacable es ver cómo se ha alargado el proyecto dos meses más de lo planeado. Como puede apreciarse, desde finales de diciembre hasta finales de febrero, los cambios realizados en el rumbo del proyecto hicieron que se tuviesen que invertir más horas de las planeadas sobre todo en la formación sobre nuevos componentes, y también en el análisis, diseño e implementación de las funcionalidades de la aplicación. En la última fase del proyecto también puede apreciarse que la fase de pruebas fue más larga de lo planeado debido a que no se esperaba que se implementasen tantas funcionalidades. Esto, sumado a la constante inserción de funcionalidades extra a lo largo del desarrollo, ha hecho exprimir al máximo las fechas de entrega del proyecto. De esta forma se ha podido crear la interfaz con el máximo número de funcionalidades posibles dentro

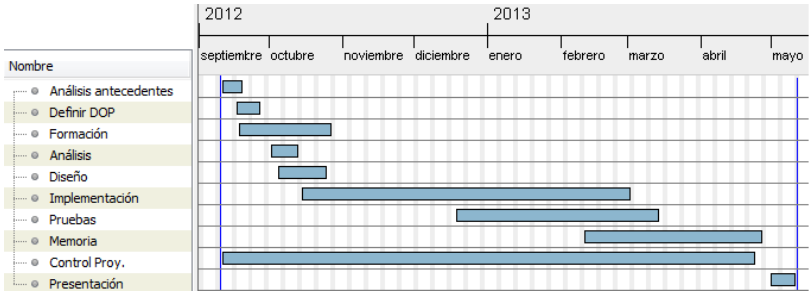


Figura V.1: Gantt estimado

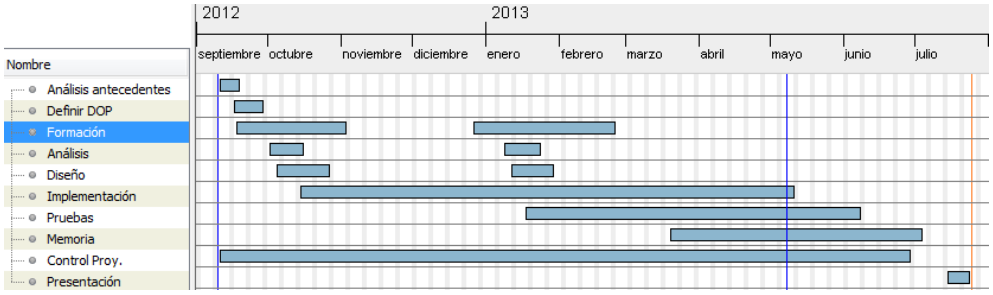


Figura V.2: Gantt final

del plazo establecido.

En el gráfico mostrado en la figura V.3 también puede apreciarse los desvios que ha habido respecto a lo planificado. Como puede verse, la implementación de nuevas funcionalidades en la interfaz ha hecho que tanto las horas de la implementación como de las pruebas crezca considerablemente.

Por último, la tabla que se muestra en la figura V.4 explica con más detalle dónde se han invertido esas horas, y también añade tareas que no estaban inicialmente planificadas. Como puede verse, el volumen total de horas ha aumentado en casi 300 horas, llegando a un total de 975.

V.2. Incidencias relevantes

A continuación se explicará con más detalle las razones de los cambios y las horas de más explicadas arriba:

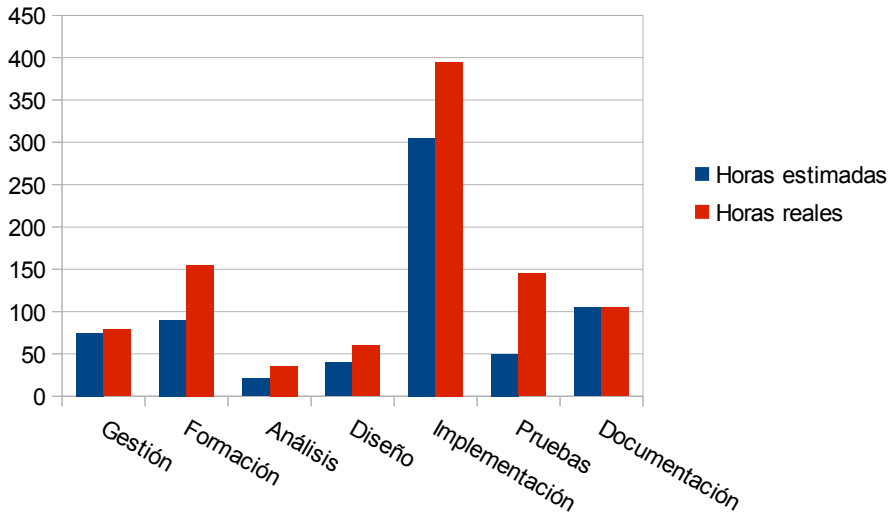


Figura V.3: Volumen de horas estimado vs reales

Categoría	Tarea	Horas reales
Gestión	Definir DOP	20+4
	Reuniones + Actas	25
	Copias de seguridad	5
	Análisis antecedentes	25
Formación	KYOTO	30
	Google Web Toolkit	25+5
	Google App Engine	5+10
	Berkeley DB + XQuery	30
	Entorno Linux	45
	Apache Tomcat	5
Desarrollo		
Análisis	Operaciones módulo minería	6
	Operaciones Nuevas	15+15
Diseño	Operaciones módulo minería	15
	Operaciones Nuevas	25+20
Implementación	Interfaz Web (GWT)	50+15
	Servidor Web (GAE)	15+15
	Integración módulo minería	40+5
	Mejorar y añadir funcionalidades	200+40
	Sustitución de GAE por Tomcat	15
Pruebas	Usabilidad interfaz	15
	Integración módulo minería	15+15
	Acceso a la app vía Web	5+20
	Operaciones nuevas	15+20
	Estabilidad componentes en linux	40
Documentación	Realizar memoria	90
	Preparar diapostivas/presentación	15
	TOTAL	**975** ~~**681**~~

Figura V.4: Horas finalmente invertidas en cada tarea

V.2.1. Migración de Windows a Linux para la instalación de BBDD

Sobre la tercera semana de diciembre se consiguieron crear las funcionalidades básicas de la interfaz, como la de parsear kybots y documentos KAF. El siguiente paso era integrar la BBDD XML Berkeley para poder avanzar con el desarrollo, ya que a partir de ahí sería necesario tener almacenados los kybots y KAFs en contenedores para su acceso. Llegados a este punto, los tutores, ya que tenían experiencia con el módulo de minería y la BBDD, conluyeron que el mejor entorno para instalar Berkeley era Linux. La razón era que nunca habían trabajado con la BBDD en Windows por lo que no se sabía si existirían incompatibilidades o cualquier otro tipo de problemas. Para no correr riesgos, por tanto, se decidió instalarla en Linux. Hasta este punto, el alumno estaba realizando todo el desarrollo sobre Windows, por lo que tuvo que migrar todos los componentes a este nuevo entorno. Supuestamente los componentes (Eclipse, *Google Web Toolkit*, etc.) usados hasta la fecha eran compatibles con Linux, por lo que en un principio no tendría que requerir mucho esfuerzo el proceso. En cambio, hubo varios problemas que hicieron que el proceso de migración durase mucho más de lo estimado. A continuación se mostrarán los factores más relevantes:

- Tras instalar varias de las distribuciones más actuales de Linux, no se consiguió instalar la base de datos Berkeley en ninguna de ellas por temas de incompatibilidad con librerías y similares. Finalmente, se logró instalar en la versión 10.10 de Ubuntu, tras haber invertido bastantes horas.

- Ya que el alumno no tenía mucha experiencia con el entorno Linux, tuvo que invertir algo de tiempo formándose para realizar todo lo necesario para la migración de los componentes y el correcto funcionamiento de la BBDD.

- Hasta que el alumno no se familiarizó completamente con el entorno nuevo, la realización de cualquier tarea tardaba más de lo planeado en un principio.

V.2.2. Migración de google App engine a Apache Tomcat

A mediados de febrero, el alumno se encontraba trabajando con los scripts en Perl usados para gestionar la BBDD Berkeley XML en el módulo de

minería. Dichos scripts debían ser invocados en el servidor, ya que ahí es donde se encuentra la BBDD. En este punto del desarrollo se estaba usando el plugin para Eclipse de *Google App Engine* para alojar la aplicación en uno de los servidores de Google. Llegados a este punto, se pudo comprobar que *Google App Engine* pone muchas restricciones a la hora de ejecutar scripts, ya sea en el lado del cliente como del servidor. Tras realizar muchas pruebas e intentos para conseguir integrarlos en el servidor, el alumno decidió que sería mejor usar otra tecnología para albergar la aplicación en un servidor. Por lo que finalmente se escogió Apache Tomcat para ello.

V.2.3. Mala planificación del esfuerzo de desarrollar las operaciones extra para la interfaz

Al realizarse la planificación se dividió la parte del desarrollo de las funcionalidades en dos grandes pilares: Las que estaban en el módulo de minería y las nuevas que se iban a crear. Las primeras, por tanto, ya estaban delimitadas y no han supuesto ningún desvío importante respecto a la planificación. Las otras, en cambio, al no tenerlas muy bien definidas en un principio, ha hecho que se haya tenido que invertir más tiempo de lo inicialemnte establecido, bien por que se hayan ido añadiendo características que no se tenían en mente al principio, o bien porque las que se tenían pensadas han sido más costosas de lo planeado. Por esto, se han tenido que invertir unas 70 horas más de lo establecido.

VI. CAPÍTULO

Conclusiones

Hasta ahora, interactuar con el módulo de minería del sistema KYOTO resultaba un tanto complejo ya que no existía ningún tipo de interfaz para sus usuarios. Cualquier acción con el módulo debía realizarse mediante comandos en el terminal, resultando poco intuitivo y complejo. Además, también se requería tener conocimientos sobre dos tipos de archivos que se utilizan en el módulo: perfiles kybot y documentos KAF. Estos archivos, estructurados en XML, pueden ser potencialmente largos y complejos de visualizar y manipular. Al igual que con el módulo, estos archivos tampoco disponían de ninguna herramienta para facilitar su uso. Por estas razones, para la mayoría de los usuarios resultaba muy tedioso y complicado la interacción con el módulo de minería.

El objetivo de este proyecto ha sido minimizar todos esos problemas creando una interfaz para poder facilitar al usuario la interacción tanto con el módulo de minería como con los archivos KAF y perfiles kybot.

Para ello, se ha creado una interfaz web que sustituye el módulo de minería, y además añade nuevas funcionalidades. La aplicación, al estar alojada en un servidor, no requiere ningún tipo de instalación en las máquinas que vaya a usarse, nada más que un navegador y conexión a internet.

Una vez se accede a la aplicación, el usuario podrá crear kybots directamente desde la interfaz o cargar uno que tenga previamente almacenado en su equipo. La interfaz, entonces, se encargará de visualizarlos en un formato mucho más fácil de interpretar y manipular. Tras esto, el usuario podrá ir editando el kybot de forma interactiva y viendo qué resultados produce al

momento. De esta manera, empezando por kybots simples, podrá ir construyendo kybots más complejos y sofisticados iteración tras iteración, hasta obtener los resultados deseados. Llegado ese momento, el usuario podrá descargar el kybot a su máquina local para su posterior uso.

Por tanto, queda claro que se ha conseguido el objetivo de facilitar al usuario la interacción con varios de los componentes del sistema KYOTO. Ahora, el usuario podrá crear y visualizar de una forma más rápida y sencilla kybots de cualquier tipo, sin tener que lidiar con muchas de las dificultades y limitaciones existentes anteriores a la creación de esta interfaz.

VI.1. Trabajo futuro

Como se estimó en un principio, existía la posibilidad de no poder integrar todas las funcionalidades pensadas, por lo que primero se implementaron las funciones básicas e importantes, y en caso de terminarlas, seguir con las opcionales. Como hemos visto, algunas de ellas han quedado pendientes, y además han surgido ideas nuevas a lo largo del desarrollo:

- Historial de acciones: la posibilidad de poder deshacer los últimos cambios realizados a los kybots sería muy interesante ya que ahorraría mucho tiempo. Los usuarios no tendrían que estar atentos de recordar el estado actual del kybot por si tiene que revertir las futuras modificaciones que haga.

- Visualización y adición de documentos KAF/Originales: Puede que haya ocasiones en las que los usuarios quieran o necesiten consultar las fuentes de información introducidas al sistema. por lo que sería interesante añadir la opción de poder visualizar tanto los documentos no procesados como los ya modificados KAF. De esta forma el usuario no se tendría que preocupar de saber la localización de dichos documentos ni de buscar editores concretos para su visualización. Por otra parte, si el usuario quisiese añadir más documentos al *Mining Module*, podría hacerlo de forma transparente mediante la interfaz.

- Gestión de usuarios: Puede que con las funcionalidades integradas hasta el momento no tuviese mucho sentido implementar esta herramienta. Pero en un futuro, cuando por ejemplo los usuarios puedan añadir, o

incluso eliminar documentos o realizar otro tipo de tareas más relevantes, sería necesario delimitar las acciones disponibles a cada usuario. Para ello, un administrador generaria cada cuenta en función de las necesidades y responsabilidades de cada usuario.

- Edición más flexible/compleja de kybots: Enriquecer aún más las funcionalidades disponibles. Por ejemplo, dar al usuario la opción de interaccionar con más atributos de los documentos KAF, o crear un sistema que evite crear kybots defectuosos o muy poco eficientes.

- Sistema integral: Bien puede ser un resumen de todas las anteriores funcionalidades. Al final lo que se pretende conseguir es un lugar desde el que se gestione todo lo relacionado al *Mining Module* (mantenimiento, carga de documentos, a la creación, modificación y consulta de cualquier elemento, etc.) sin que el usuario tenga que entender cómo funciona cada uno de sus componentes.

- Implantación de la aplicación en entorno real: De momento la aplicación está instalada en el ordenador del alumno. De todas formas, la intención será implantarlo en un entorno donde los usuarios de KYOTO tengan acceso a la aplicación.

Por otra parte, se podría trabajar en optimizar todo lo implementado: Hacer la interfaz aún más fácil de entender para los usuarios con escasos conocimientos informáticos, y también se tendría que estudiar cómo mejorar el rendimiento de la aplicación para que funcione sin problemas en cualquier tipo de navegador y en equipos de escasos recursos o conexiones lentas.

VI.2. Opinión personal y agradecimientos

Para terminar, debo decir que estoy muy satisfecho por dos razones:

Estoy satisfecho haber elegido este proyecto, ya que aparte de formarme en todo lo relativo al sistema KYOTO, he tenido que aprender a manejar multidud herramientas de software que han hecho falta para sacar este proyecto adelante. Por tanto, creo que en estos meses he adquirido bastantes conocimientos técnicos que seguramente tendré que usar en un futuro cercano. Además, el hecho de desarrollar una herramienta para KYOTO, un proyecto real, ha hecho que aprenda a afrontar los problemas que pueden

surgir en cualquier futuro proyecto similar. Esto, también ha hecho que la motivación durante el desarrollo sea mucho mayor, ya que la herramienta creada podrá ser de utilidad para los usuarios de KYOTO.

Y por último, estoy satisfecho por haber trabajado con Aitor y German ya que a pesar de las dificultades mencionadas, ellos me han ayudado siempre que no sabía por dónde seguir y siempre me han dado sugerencias para mejorar cada aspecto de la aplicación. Por ello, les doy las gracias por todo su apoyo durante estos meses.

VII. CAPÍTULO

Anexos

VII.1. Manual de usuario de instalación y uso de la interfaz

VII.1.1. Indroducción

En este manual se explicará por una parte cómo instalar y configurar los elementos necesarios para poder usar la interfaz que servirá para visualizar y editar kybots. Por otra parte, también se mostrará cómo acceder a dicha interfaz y cómo usar cada una de sus funcionalidades.

VII.1.2. Instalación y configuración

1. Para el correcto funcionamiento de la BBDD Berkeley y sus librerías, se tendrá que instalar la versión 10.10 del sistema operativo Ubuntu[1].

2. Descargar Apache Tomcat[2] y proceder a su instalación y configuración:

   a) Escoger un directorio para su instalación.

   b) Configurar las variable *CATALINA_HOME* y *PATH*, indicando el directorio raíz donde se tenía instalado Tomcat.

[1] http://old-releases.ubuntu.com/releases/maverick/
[2] http://tomcat.apache.org/download-70.cgi

c) Verificar la correcta instalación invocando el comando *catalina.sh*, el cual inicia el servidor Tomcat.

d) Una vez invocado el comando, abrir una ventana del navegador web y visitar la URL *http://localhost:8080* para comprobar que sale la página de documentación de Tomcat.

3. Ubicar el archivo *kybotbyexample.war* en la carpeta *webapps* dentro del directorio donde se haya instalado Tomcat.

4. Ubicar carpeta *mining_module* dentro del directorio de instalación de Tomcat. Esta carpeta contendrá los archivos auxiliares necesarios para el funcionamiento de la aplicación. Para añadir archivos KAF al sistema, se tendrán que situar en la carpeta *docs_en* y arrancar la aplicación.

5. Ejecutar desde el terminal el script *startup.sh* de Tomcat para arrancar el servidor. Para comprobar que todo funciona correctamente se debe entrar al navegador y poner la dirección
localhost:8080\kybotbyexample y debería cargarse la interfaz.

Una vez realizados estos pasos, la interfaz estaría lista para usarse en modo local. Para que esté disponible a través de internet, se tendría que conseguir un servicio de *Hosting* para la aplicación, a elección del usuario.

VII.1.3. Acceso y uso de la aplicación

1. Para acceder a la interfaz el usuario deberá introducir la siguiente dirección en su navegador:
dirección IP de la máquina: puerto escogido\kybotbyexample.html. Un ejemplo, usando *localhost*, sería *127.0.0.1:38684\KybotByExample.html*. Una vez cargada la página se tendrá que ver algo similar a lo que muestra la figura VII.1. En ella se muestra la interfaz nada más ser arrancada, sin mostrar ningún tipo de información. Como puede verse, se divide en tres módulos principales. La esquina superior derecha es el panel donde se mostrará el kybot que se esté editando. El borde superior del panel mostrará el nombre del kybot, y a su derecha pueden verse las distintas operaciones que pueden hacerse con los kybots: *Guardar*, *Nuevo*, *Procesar* y *Cargar*. Por otro lado, en la esquina inferior derecha tenemos el panel de resultados. Aquí aparecerán los resultados obtenidos cuando se ejecutan los kybots. Pinchando sobre los botones *raw* o

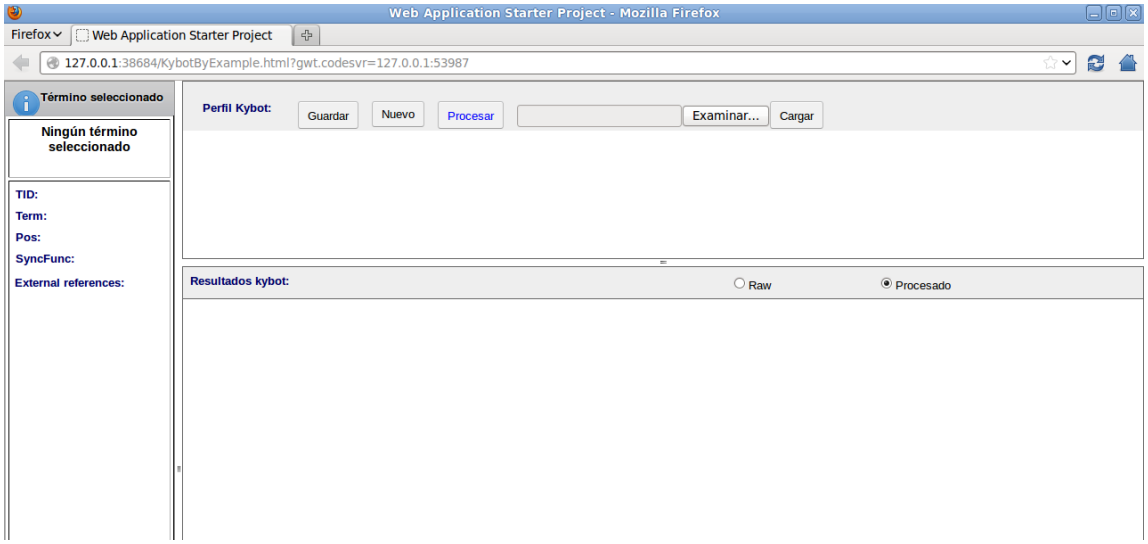


Figura VII.1: Ascpecto de la interfaz nada más ser arrancada

procesado, el usuario podrá elegir el tipo de salida que quiere ver. Por último, la columna que aparece en la izquierda sirve para mostrar información acerca de los terminos que se vayan seleccionando del panel de resultados. Ya que el usuario podrá editar el kybot usando estos resultados, este panel informativo le servirá de ayuda para determinar si los términos que va seleccionando tienen las características deseadas.

2. Una vez se tenga la interfaz en marcha, podrán hacerse las siguientes operaciones:

   a) Crear un kybot: Para ello, pulsaremos en el botón *Nuevo* y nos saldrá un *popup* como el que aparece en la figura VII.2. El usuario deberá indicar el nombre del kybot, y definir la primera variable que tendrá. Para ello, pondrá un nombre a la variable, y tendrá que rellenar al menos el campo del *lemma* o del *part of speech*(pos). Una vez rellenados dichos campos se deberá pulsar la tecla *crear kybot*, y se mostrará en el panel del kybot nuestra creación al igual que en la figura VII.3

   b) Cargar un kybot: Para ello, tendremos que pulsar el botón *Examinar*, seleccionar uno de los archivos XML que contenga un kybot, y pulsar en el botón *cargar*. El kybot seleccionado se mostrará en

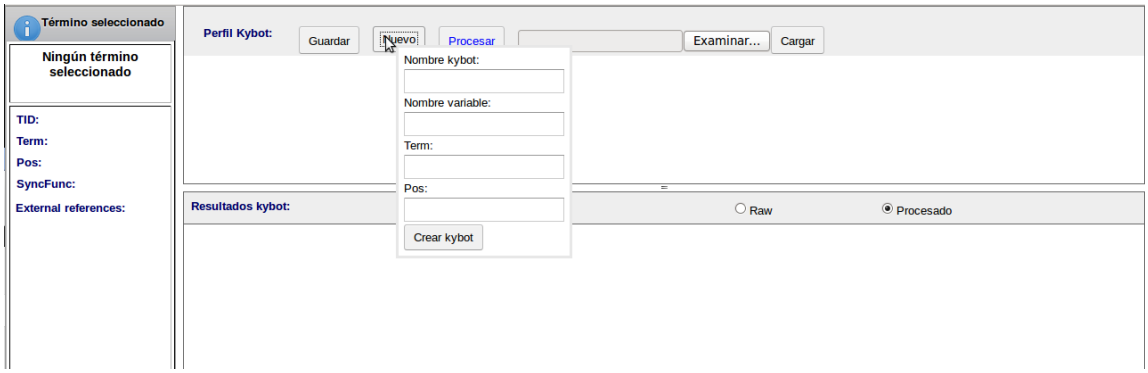


Figura VII.2: *Popup* en el que se introducirán los datos que se desean

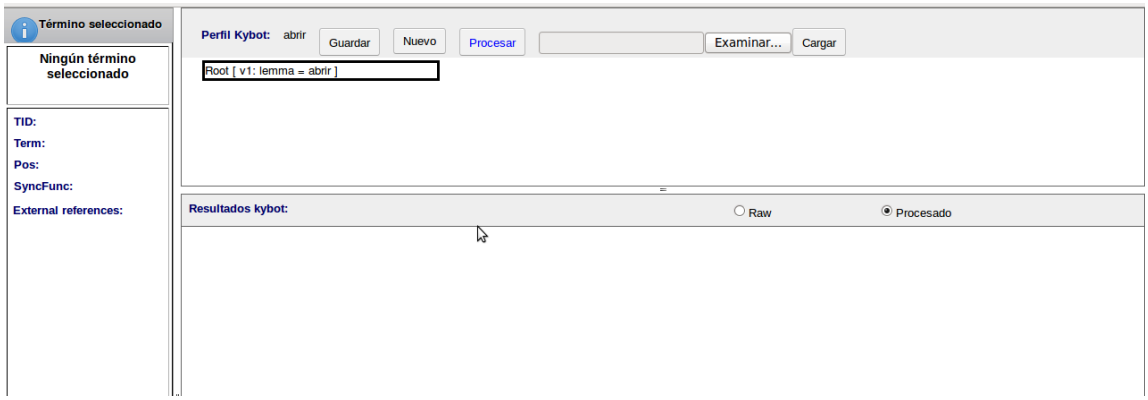


Figura VII.3: Kybot una vez rellenado el *popup*

pantalla.

Llegados a este punto, de una forma u otra aparecerá un kybot en pantalla. A continuación se explicará cómo se representan los kybots en la aplicación mediante el ejemplo de la figura VII.4.
La aplicación representa los kybots siguiendo la estructura de un árbol, y cada variable será uno de los elementos de árbol. La variable que haya sido elegida como *root* (raíz), será la primera en mostrarse. Además, llevará el identificardor *Root* antes de mostrar la variable. El resto de variables tendrán $<-$ o bien $->$ antes de mostrar la variable, lo cual indicará si la variable va antes o después de su padre. En el caso de que estas variables tengan la restriccion *Immediate*, es decir, que no pueda ir ninguna otra variable en medio, los signos empleados serán $<<--$ y $-->>$. Para entenderlo mejor, en el dibujo puede verse que la variable *e*, con el icono $->$, es el hijo de la variable *b*, lo que querrá decir que *e* va después de *b*.
Una vez establecidas las dependencias entre los nodos del árbol, o dicho de otra forma, las variables del kybot, vendrá la información sobre cada variable, la cual estará entre los dos corchetes. Aquí se indicará, en caso de que haya sido definido, el *pos*, *lemma* y *external reference*.

3. Una vez se tenga un kybot en pantalla, ya sea creado o subido, se podrán relizar las siguientes acciones:

   a) Guardar kybot: Para ello se pulsará el boton *Guardar*, y en breves momentos comenzará la descarga del archivo XML que contenga el kybot. Por supuesto, este archivo podrá subirse al sistema usando la opción de *cargar kybot* mencionada anteriormente. De esta forma el usuario podrá trabajar con un kybot en distintas sesiones.

   b) Procesar Kybot: Una vez pulsado el botón *Procesar*, el panel inferior derecho se rellenará con los resultados proceados por el kybot. Si está seleccionada la opción *Raw*, los resultados saldrán tal y como salían en el módulo de minería (figura VII.4), en cambio, si se escoge la opción *Procesado*, los resultados saldrán como se muestra en la figura VII.5 y los términos podrán usarse para editar el kybot como veremos en el punto *d)*.

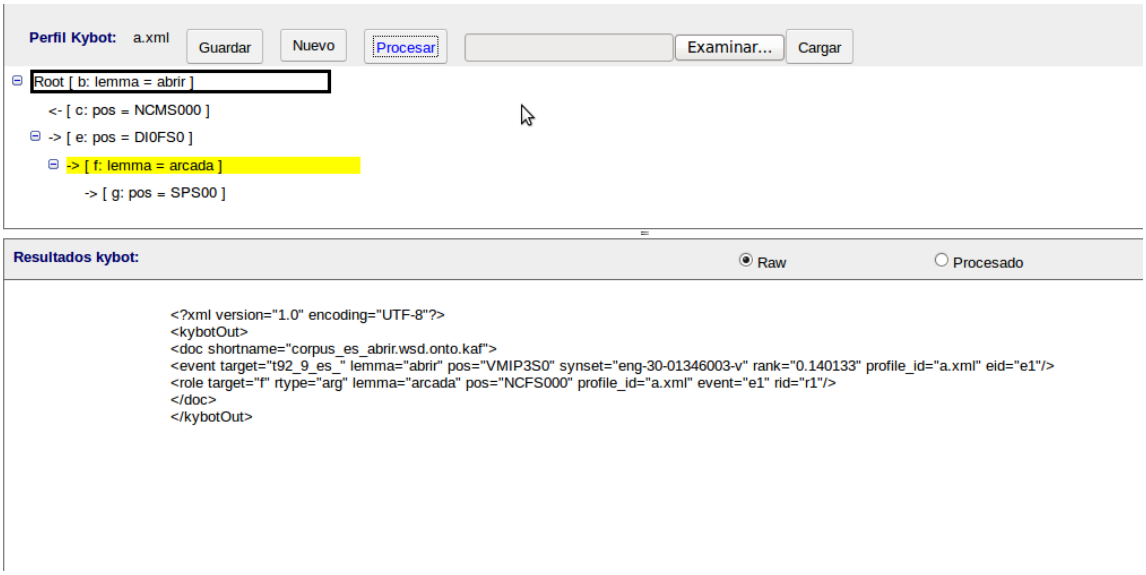


Figura VII.4: Formato de la salida si se ha elegido la opción *Raw*

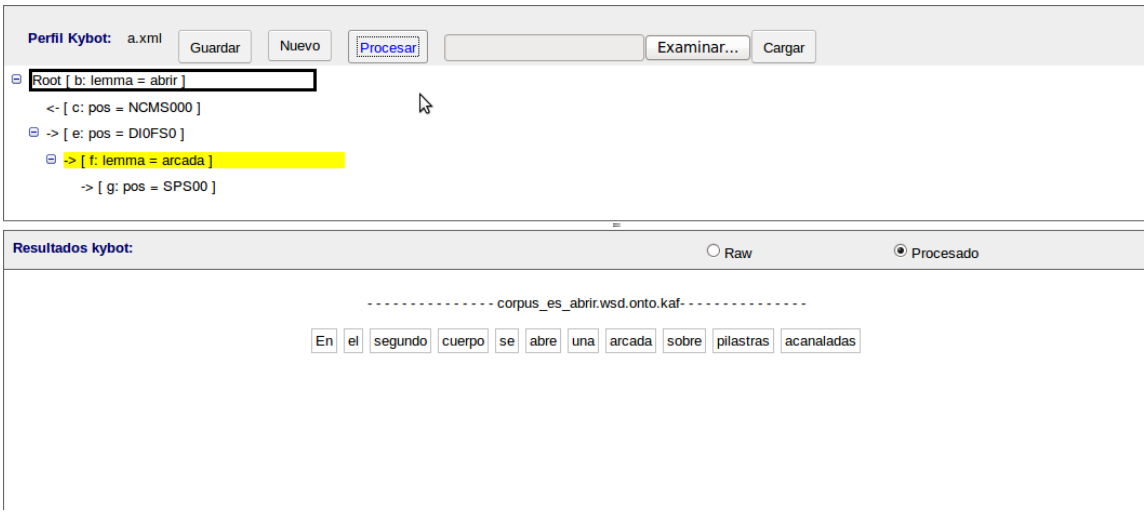


Figura VII.5: Formato de la salida si se ha elegido la opción *Procesado*

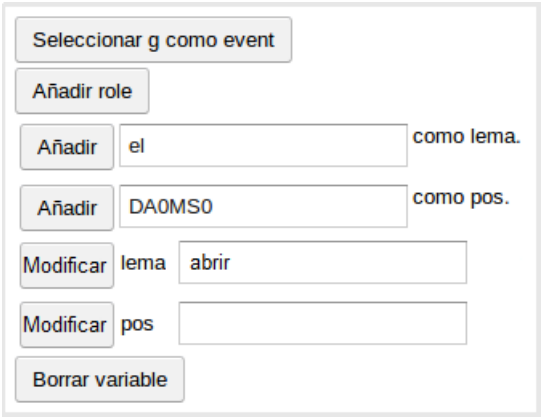


Figura VII.6: *Popup* mostrado a la hora de pinchar en un componente del árbol para proceder a su edición.

c) Volver a crear o cargar un kybot, al igual que en el paso 2. El usuario debe tener en cuenta que perderá los datos del kybot actual, si no guarda el fichero.

d) Editar Kybot: El usuario podrá ir modificando la estructura del kybot, e ir viendo al momento todo los cambios de forma visual. Para realizar cualquier cambio, se deberá pinchar en cualquiera de los componentes, es decir, variables que forman la estructura del kybot, y aparecerá un *popup* similar al de la imagen VII.6. El usuario podrá realizar los siguientes cambios al kybot:

- Modificar la información sobre la variable seleccionada: Por ejemplo, si se había definido una variable que contuviese *abrir* como lema, podría cambiarse *abrir* por otro nuevo valor. Esto también se aplica al *pos* y a las *external references*. Por otro lado, también se podría modificar el valor *Immediate* previamente definido en la variable.
- Borrar la variable seleccionada junto con todos los hijos. Es decir, borrar la variable junto con todas las que dependan de ésta.
- Añadir una nueva variable. Para ello pincharemos sobre una de las variables ya creadas del kybot. Tras ello se definirá qué quiere añadirse: *lemma*, *pos*, *external reference*...y por último se determinará si la variable nueva irá antes o después de la

variable en la que haya pinchado, y si tendrá el atributo *Immediate*. En este caso puede ser una variable introducida por el usuario, o una previamente seleccionada de los resultados al ejecutar un kybot con la opción *Procesado*.

- Si la variable seleccionada no es *event* ni *role*, se podrá seleccionar para que sea el nuevo *event*. Siempre habrá una única variable con ese atributo, y ésta se identificará con un marco negro rodeando la variable, como muestra la figura VII.4. Por lo tanto, si selecciona una nueva variable, la anterior que tuviese el atributo *event* lo perderá. No habrá ninguna variable que tenga los dos atributos a la vez.
- Ocurre lo mismo con *role*. La única diferencia es que puede haber más de una variable con *role*. La única restricción será que una variable no puede ser *role* y *event* a la vez. Las variables con *role* se identificarán con un fondo amarillo, como muestra la figura VII.4. Por otro lado, habrá que asignarle un nombre para rellenar el atributo *arg*. Este nombre se mostrará si se ejecuta el kybot con la opción *Raw*.
- Si la variable ha sido definida como role previamente, borrar dicho atributo.

Bibliografía

Kafinspector. URL `http://weblab.iit.cnr.it/kafdebugger/`.

Bosma W.E., Vossen P., Soroa A., Rigau G., Tesconi M., Marchetti A., Monachini M., eta Aliprandi C. Kaf: a generic semantic annotation format. *Proceedings of the GL2009 Workshop on Semantic Annotation. Pisa, Italy*, 2009.

German R. eta Eneko A. Storyboard: to mine by example for building kybots. *In project: Knowledge Yielding Ontologies for Transition-based Organization*, 2007.

Piek V., Eneko A., German R., , eta Aitor S. Kyoto: a knowledge-rich approach to the interoperable mining of events from text. *Book chapter in: New Trends of Research in Ontologies and Lexical Resources*, 2013.

Rigau G. Kyoto demo. URL `http://ixa2.si.ehu.es/demokaf/demokaf.pl`.

Rigau G. Kybots, knowledge yielding robots. *The First KYOTO Workshop. Amsterdam, Netherlands*, 2009.

Rigau G., Soroa A., Vossen P., Zapirain B., Laparra E., Agirre E., Gojenola K., Casillas A., eta de Illarraza Kike Fernández A.D. Fact miners revised. *In project: Knowledge Yielding Ontologies for Transition-based Organization*, 2010.

Vogel L. Gwt tutorial, September 2011. URL `http://www.vogella.com/articles/GWT/article.html`.

Vossen P., Soroa A., Zapirain B., eta Rigau G. Cross-lingual event-mining using wordnet as a shared knowledge interface. *Proceedings of the 6th Global WordNet Conference (GWC'12)*, 2012.