Universidad del País Vasco

Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Generación de preguntas sobre un texto

Autor

Daniel Alfaro Paitamala

Directora

Montserrat Maritxalar Anglada

Febrero del 2016

Agradecimientos

Quisiera empezar esta lista expresando mi agradecimiento a mi directora del proyecto, Montse Maritxalar, por dedicarme su tiempo en encaminarme para sacar este proyecto adelante, por ayudarme en la búsqueda de soluciones, y sobre todo por la confianza depositada desde el primer momento. Por esto y por muchas otras cosas más, muchas gracias.

Muchas gracias a Rodrigo Agerri por ilustrarme en el manejo de las herramientas IXA-Pipes.

También quiero agradecer a Anartz Recalde y José Alfaro, por tomarse la molestia de evaluar el generador de preguntas.

Gracias a mis padres por estar siempre presentes a lo largo de mi formación, por darme ánimos y apoyarme de manera incondicional.

Gracias a los profesores que a lo largo de la carrera han contribuido en mi desarrollo como profesional y que sin sus enseñanzas no habría sido esto posible.

Gracias a los compañeros de clase con los cuales he compartido grandes momentos dentro y fuera de la universidad, puede que la distancia nos separe a muchos, pero el recuerdo siempre permanecerá.

Por último me gustaría agradecer a todas las personas que han seguido mi desarrollo a lo largo de la carrera, que siempre han mostrado un interés en que todo me vaya bien. A todos mis amigos y conocidos, muchas gracias.

Resumen

Este Proyecto tiene como objetivo principal la generación de preguntas a partir de un texto, las preguntas obtenidas serán a nivel de sentencia, para ello se ha creado dos programas, cada uno de estos programas es capaz de generar preguntas, pero cada una de estas generaciones está basada en distintos análisis:

- Generador de preguntas a partir de un análisis morfosintáctico.
- Generador de preguntas a partir de un análisis basado en dependencias sintácticas.

Ambos generadores parten de una arquitectura similar que está formada por una selección del objetivo, una selección del tipo de pregunta, y una construcción de la pregunta. En estas etapas se han desarrollado diversos algoritmos, en general cabe destacar los relacionados con la selección del tipo de pregunta.

Para probar si estos programas y los algoritmos que los acompañan son fiables se ha desarrollado también una evaluación sobre estos.

Palabras Clave: PNL, NLP, computación, Ixa-pipes, Generación de Preguntas, Analizador morfosintáctico, Analizador basado en dependencias.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Durante el transcurso de los últimos años el Procesamiento del Lenguaje Natural ha sido y es un área en continuo desarrollo, el PLN hace referencia a la interacción entre las computadoras y el lenguaje humano, la comunicación entre ambos está diseñada mediante mecanismos eficaces que se puedan realizar por medio de programas. El interés que suscita tiene su origen en 1950, ya desde esa época Alan Turing publicó Computing machinery and intelligence dando forma e introduciendo lo que hoy conocemos como el test de turing. Si bien es cierto que hoy en día hemos llegado a nuevos niveles en el campo del PLN, no está de más saber que el proceso ha sido más lento de lo esperado, y decimos más lento no por el interés, sino porque los primeros augurios sobre la evolución del PLN no fueron del todo acertados, no obstante en términos generales los resultados han sido más que satisfactorios, y ¿por qué el proceso ha sido más lento?, el PLN cuenta con tres grandes dificultades:

- Ambigüedad.- El lenguaje natural es inherentemente ambiguo a diferentes niveles (léxico, referencial, estructural y pragmático).
- Detección de separación entre palabras.- En la lengua hablada no se suele hacer pausas entre palabra y palabra. El lugar en el que se debe separar las palabras a menudo depende de cuál es la posibilidad que mantenga un sentido lógico tanto gramatical como contextual.
- Recepción imperfecta de datos.- Acentos extranjeros, regionalismos o dificultades en la producción del habla, errores de mecanografiado o expresiones no gramaticales, errores en la lectura de textos.

Estas dificultades que no fueron detectadas en su momento han sido la causa del lento desarrollo que ha tenido el PLN. En la actualidad estos puntos mencionados se encuentran en cualquier trabajo vinculado con el PLN. En la última década se ha experimentado un gran crecimiento en los trabajos relacionados con el PLN, con el crecimiento de estos trabajos también han crecido el número de objetivos.

Uno de los tantos objetivos del PLN es el de desarrollar diversos generadores de preguntas a partir de una sentencia o de un texto completo, estos generadores están sujetos a las normas gramaticales de los distintos idiomas en los que puede estar escrito. Estos textos escritos, gracias a internet, pueden ser accesibles a través de una búsqueda en nuestro navegador, los textos son muy variados, en algunos casos con faltas de ortografía, errores que pesan a la hora de trabajar con ellos. Por suerte la aparición de diversas plataformas, de las que cabe destacar Wikipedia, dan una mayor credibilidad a la hora de trabajar con estos, tener en una web abierta al público una gran diversidad de textos en distintos idiomas y que por lo general no tienen faltas de ortografía es desde mi punto de vista un bien preciado.

Volviendo al tema central, el generador de preguntas es una aplicación útil en diversos campos lingüísticos, un generador será capaz de ahorrar tiempo en la lectura de un texto y ser capaz de obtener todas las preguntas que se puedan hacer sobre dicho texto. La

precisión, la exactitud, la relevancia de la pregunta vendrán determinados por los diversos algoritmos desarrollados en la implementación.

Un paso previo al inicio de abordar el proyecto fue la lectura de distintos artículos que incluían trabajos e investigaciones relacionadas con el proyecto. Gracias a estos se obtuvieron diversas ideas sobre cómo evaluar el generador que se propone en este proyecto. De esta forma se podrá saber si el generador es eficiente, o que tan eficiente puede llegar a ser.

A lo largo de la memoria se explicarán todos los pasos dados, las dificultades tenidas, los estudios realizados, los algoritmos creados, los resultados obtenidos, etc.., sobre lo que conlleva generar una aplicación de esta magnitud.

1.1. **Objetivo**

Al inicio del proyecto durante la creación del alcance se propuso como objetivo la generación de preguntas a partir de un mapa conceptual. No obstante después de evaluar la dificultad que este suponía se volvieron a redefinir los objetivos y alcances. Sin embargo en las conclusiones hay un apartado que habla sobre todo lo referente que se llegó a desarrollar, sobre todo la parte de diseño, y se propone como una posible mejora.

A continuación los dos tipos de objetivos que tiene este proyecto:

- Objetivo general
    - Crear una aplicación que permita generar preguntas a partir de un análisis morfosintáctico.
    - Crear una aplicación que permita generar preguntas a partir de un análisis basado en dependencias.

- Objetivos secundarios
    - Plantear un diseño básico sobre la generación de preguntas basado en un mapa conceptual.
    - Aprender a diseñar soluciones algorítmicas para la resolución de problemas.
    - Aprender y usar herramientas del PLN
    - Dotar de conocimiento a los sistemas informáticos.
    - Resolver problemas de forma inteligente

1.2. **Alcance**

- Aplicación desarrollada en lenguaje JAVA usando la plataforma Eclipse.
    - Versión: Luna Service Release 1 (4.4.1).

- El generador solo generará preguntas de textos en español a nivel de sentencia.

- Generación de preguntas en base a los distintos tipos de análisis:
    - Morfo-sintáctico: Se deberá se capaz de generar preguntas a partir de un análisis morfo-sintáctico en el apartado 2.2 se explica en que consiste.

    - Analizador basado en dependencias: Se deberá ser capaz de generar preguntas a partir de un analizador basado en dependencias, en el apartado 2.3 se explica en que consiste.
- El dominio de la aplicación será de tipo histórico, enfocado en el tema de guerras.

- La generación de preguntas será del tipo conceptual.
    - Qué:
      "¿Qué ocurrió en el año 353 entre las fuerzas de Constancio II y las fuerzas del usurpador Magnencio?"

    - Quién / Quiénes:
      "¿Quién era un hombre sabio y capaz, con un espíritu inquieto e insaciable?"

    - Cuándo:
      "¿Cuándo iniciaron las hostilidades los almorávides?"

    - Dónde:
      "¿Dónde aconteció la gran batalla de Abrito?"

1.3. **Contenido**

El documento está estructurado de la siguiente manera:

- El capítulo 2 nos proporcionará distintos conceptos básicos los cuales nos ayudarán a entender con una mayor claridad la información en este documento.
- El capítulo 3 presenta el estado del arte y con ello un estudio realizado a partir de los últimos avances en el tema de la generación de preguntas.
- El capítulo 4 está centrado en describir las herramientas que se han usado a lo largo del proyecto.
- El capítulo 5 describe los pasos dados para desarrollar los dos generadores de preguntas, en este apartado se explica las distintas decisiones tomadas, la arquitectura que tienen, la selección del corpus, la creación de estructuras para poder trabajar con los textos, también se informará de algunos problemas encontrados y en algunos casos las soluciones propuestas.
- El capítulo 6 nos informará de las distintas evaluaciones realizadas con los generadores.
- El capítulo 7 es un informe sobre cómo se ha gestionado el proyecto de principio a fin, incluye el EDT (estructura de descomposición del trabajo), costes, gráfico de Gantt, etc.
- Por último en capítulo 8 se presentan las conclusiones finales y posibles mejoras.

Capítulo 2

Conceptos básicos

2.1. Definición de Procesamiento del Lenguaje Natural (PLN)

Es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano, para ello se diseñan mecanismos para comunicarse que sean eficientes computacionalmente.

Algunos de los objetivos más destacables del PLN son (1):

- Interfaces en lenguaje natural: Una de las metas consiste en poder dar órdenes en el mismo lenguaje a todos los ordenadores.
- Procesamiento de textos: se abortan temas como el de la recuperación de información, extracción de datos significativos, la elaboración de resúmenes, etc.
- Traducción automática: El objetivo original del PLN y también en el que se han conseguido mayores resultados.

2.1.1. Evolución del PLN

Los primeros datos sobre aplicaciones basados en PLN se remontan al año 1948, y nos cuentan como en Londres se desarrolló una aplicación de búsqueda en un sistema basado en un diccionario. Al año siguiente los americanos se interesaron en este tipo de aplicaciones dándole una nueva utilidad durante la Segunda Guerra Mundial. (2)

En 1954 apareció el primer traductor automático por parte de la Universidad de Georgetown, la aplicación podía traducir automáticamente más de sesenta oraciones del ruso al inglés. Los autores declararon que en un par de años la traducción automática sería un problema resuelto.

Durante los años sesenta las técnicas se centraron en la comprensión del lenguaje, y se trabajó con diversas técnicas de análisis sintáctico. (3)

En los años setenta la influencia de trabajos de inteligencia artificial influye de manera decisiva, centrando su interés en la representación del significado. Como resultado se construyó el primer sistema de preguntas-respuestas basado en lenguaje natural.

Con el paso de los años los programas se han vuelto más fiables, y han ido apareciendo gramáticas orientadas a un tratamiento computacional lo cual hace que crezca considerablemente la programación lógica.

En los últimos años se han incorporado técnicas basadas en la estadística y se desarrollan distintos formalismos para tratar la información léxica, un buen ejemplo de ello son las webs semánticas.

Se ha realizado una búsqueda en el sitio web ieee explore con las palabras clave <<Natural Language Processing>>, y se ha elaborado la siguiente gráfica lineal para saber el grado de crecimiento que ha tenido desde 1954 hasta 2015.

Figura 2-1: Resultados de búsqueda en el sitio web iee explore (4) con la palabra clave <<Natural Language Processing>>

2.1.2. Aplicaciones del PLN

A continuación se pondrán como ejemplo un par de aplicaciones del PLN, la primera es básicamente el que se está desarrollando en este proyecto, los otros dos ejemplos son aplicaciones al margen del proyecto pero que forman parte de la familia del PLN.

- Aplicación concreta relacionada con el proyecto
    - **Generador de Preguntas**

      El generador de preguntas (Generating questions en inglés) es una aplicación que a partir de una sentencia, una frase o un texto, es capaz de generar preguntas con sentido sobre estas.

      De momento se han desarrollado diversos generadores, cada uno destinado exclusivamente a un idioma.

- Aplicaciones generales
    - **Web-semántica**

      El objetivo es mejorar Internet ampliando la operatividad entre los sistemas informáticos usando "agentes inteligentes". Los agentes

inteligentes son programas en las computadoras que buscan información sin operadores humanos.

- **Traductores automáticos**

  Su objetivo es la traducción de un idioma a otro manteniendo la coherencia en la frase.

2.2. Definición de Analizador Morfosintáctico

Un analizador morfo-sintáctico realiza un análisis morfológico y un análisis sintáctico.

Ejemplo:

La batalla fue dura.

Una forma de mostrar el análisis de manera poco orientativa sería la siguiente:

(S (SN (DA0FS0 ***La***) (GRUP.NOM (NCFS000 ***batalla***))) (GRUP.VERB (VSIS3S0 ***fue***)) (SA (GRUP.A (AQ0FS0 ***dura***)))) (FP .)

A continuación se muestra un gráfico en forma de árbol que permitirá mostrar este mismo análisis pero de una forma más clara.

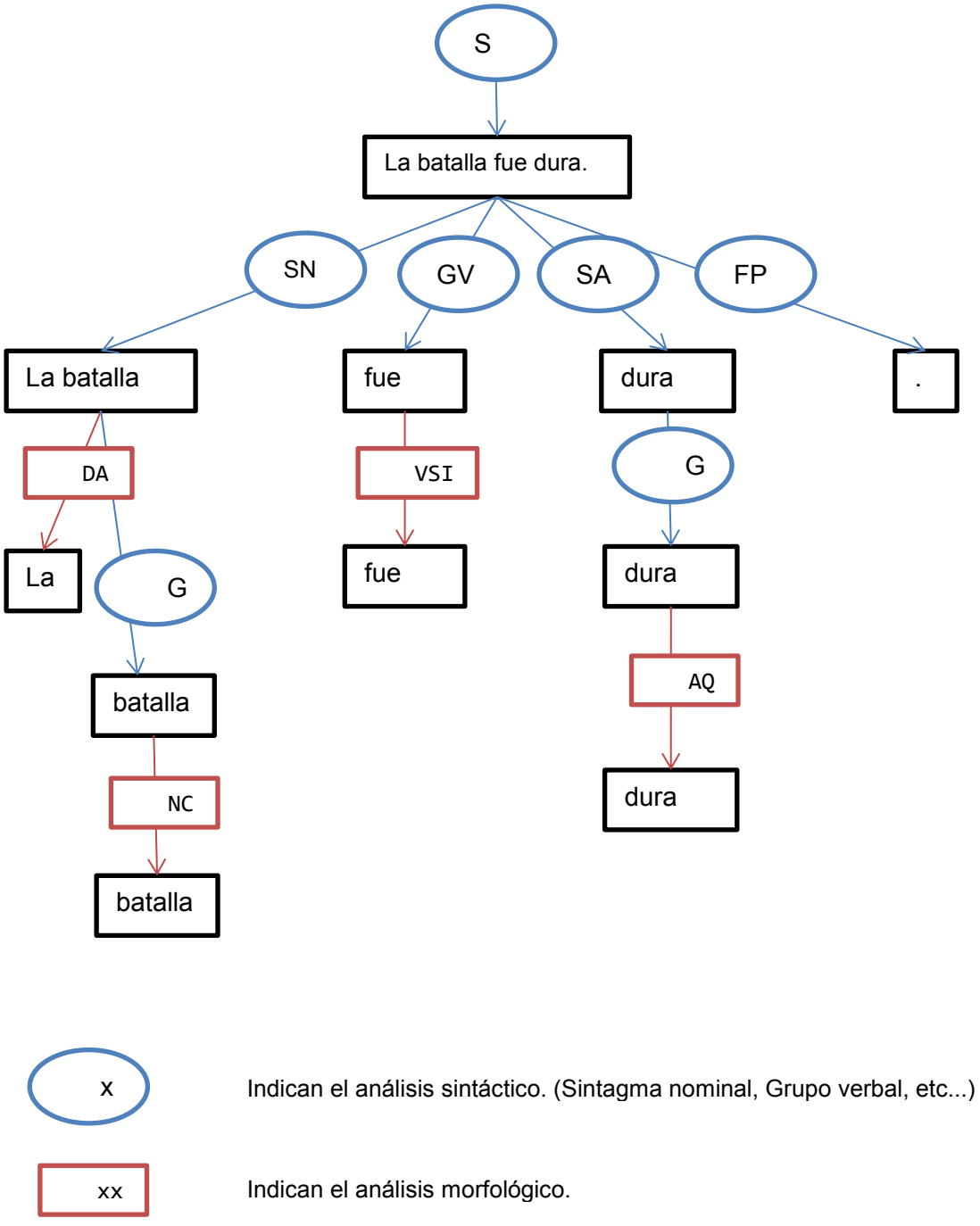


Figura 2-2: Análisis morfo-sintáctico.

En el apartado 4.1.2 se puede ver que indica cada letra del análisis morfológico.

2.3. **Definición de Analizador basado en dependencias.**

El analizador basando en dependencias es un analizador sintáctico con un enfoque distinto el cual consiste en trabajar con las relaciones existentes entre las palabras de la frase.

Las dependencias pueden estar anotadas (tener un nombre).

El conjunto de dependencias de una frase es más manejable en forma de árbol de dependencias.

Ejemplo:

Para realizar este ejemplo usaremos la misma frase que en el analizador morfosintáctico:

La batalla fue dura.

A continuación se muestran la lista de dependencias de la manera que nos muestra la salida del sistema.

```
ITERACION : 0 -----spec(batalla, la)
                  ----- la ---Tipo: D
ITERACION : 1 -----suj(fue, batalla)
                  ----- batalla ---Tipo: N
ITERACION : 2 -----atr(fue, dura)
                  ----- dura ---Tipo: G
ITERACION : 3 -----f(fue, .)
                  ----- . ---Tipo: O
```

Esto se puede traducir de manera gráfica a la siguiente figura:

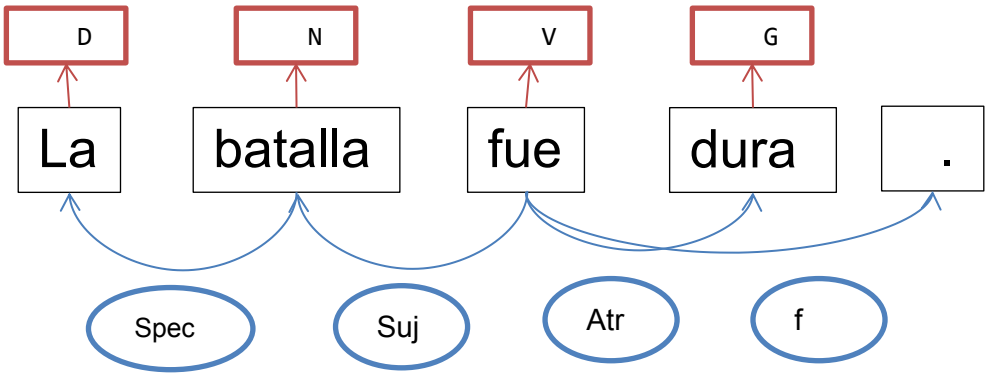


Figura 2-3: Dependencias

A continuación se muestra este mismo análisis en forma de árbol.

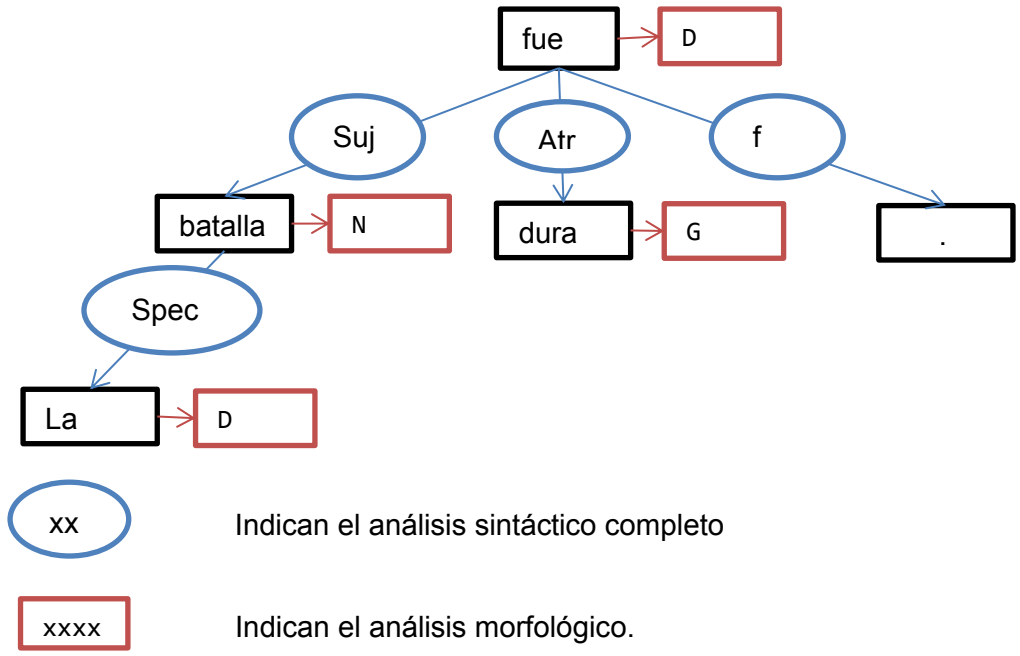


Figura 2-4: árbol de dependencias

2.4. Definición de palabra interrogativa

En lingüística, una palabra interrogativa o qu-palabra (del inglés wh-word) es una palabra funcional usada para formular interrogativas parciales (5).

Las palabras interrogativas en español se agrupan en 4 bloques:

	Qué	Cuánto	Quién	Cuál	Dónde	Cómo	Cuándo	Preposición + interrogativo
Determinantes Interrogativos	◆	◆						
Pronombres interrogativos	◆		◆	◆				
Proformas interrogativas		◆			◆	◆	◆	
Formas compuestas								◆

Tabla 2-1: Palabras interrogativas

Ejemplos de interrogativas:

1. Determinantes interrogativos ("adjetivos interrogativos")

   Qué: *¿**Qué batallas** se libraron?*

   Cuánto, -a, -os, -as: *¿**Cuántas guerras** se lucharon ahí?*

2. Pronombres interrogativos

   Quién (personal): *¿**Quién** dijo eso?*

   Qué (general, indefinido): ***Qué** batalla aconteció?*

   Cuál (general, definido): *¿**Cuál** falta?*

3. Proformas interrogativas ("adverbios interrogativos")

   Dónde (lugar): *¿**Dónde** sucedió?*

   Cuándo (tiempo): *¿**Cuándo** sucedió?*

   Cómo (manera): *¿**Cómo** te vas?*

   Cuánto (cantidad): *¿**Cuánto** es mucho?*

4. Formas compuestas

   Preposición + interrogativo: *¿**Con quién** lucho?*, *¿**Desde qué** lugar?*, *¿**Hasta cuándo** sucedió?*

Capítulo 3

Estado del arte

A continuación se presenta una síntesis de aquellos artículos relacionados en su totalidad con la generación de preguntas, estos hablan sobre las últimas técnicas desarrolladas por distintos grupos de investigación.

Por ello se muestra una síntesis de aquellos artículos leídos que han tenido un gran impacto en la elaboración de este proyecto.

3.1. **Question Generation Shared Task Evaluation Challenge (6)**

Es un desafío que reúne diversos generadores de preguntas con el fin de evaluarlos. Los generadores de pregunta podían especializarse en las preguntas a partir de un párrafo (A) o preguntas a partir de una sentencia (B).

Estos generadores A y B, se evaluaban de manera distinta en base a distintos criterios.

- Generadores A – preguntas sobre un párrafo.
  Se reta a generar 6 preguntas de un párrafo de entrada, estas estarán en tres niveles de especificidad:

  - Una pregunta de carácter general, de todo el párrafo.
  - Dos preguntas en base a una o dos sentencias.
  - Tres preguntas a partir de una frase.

  Los criterios de evaluación, se evalúan con un valor de uno en caso de ser correcta o cero en caso contrario, y estos serían:

  - Especificidad
  - Sintaxis
  - Semántica
  - Corrección del tipo de pregunta.
  - Diversidad

  Cada pregunta es evaluada en base a estos criterios.

- Generadores B – preguntas sobre una sentencia.
  Se reta a generar preguntas a partir de una sentencia, clasificando cuantos tipos de preguntas se pueden generar. Estos tipos de pregunta serían:

  - ¿Quién?, describe un personaje.
  - ¿Dónde?, describe un espacio físico.
  - ¿Cuándo?, describe un espacio temporal
  - ¿Cuál?, describe una categoría.

- ¿Qué?, describe una específica entidad.
- ¿Por qué?, describe una causalidad.
- ¿Cuánto?, describe una cantidad.

Los criterios de evaluación son los siguientes:

- Relevancia
- Tipo de pregunta.
- Exactitud sintáctica y fluidez.
- Ambigüedad.
- Variedad.

La puntuación está asociada con una escala que va de 1 a N (2, 3, 4), siendo 1 mejor y N peor.

3.2. **Question Generation for French: collating parsers and paraphrasing questions (7)**

El artículo presenta un sistema de generación de preguntas para el idioma Francés y recalca desde el inicio que los métodos que se proponen para el lenguaje Inglés no son equivalentes a los sistemas de lenguaje Francés.

Mientras que en inglés estos serían los pasos para la generación de preguntas:

- Realizar un análisis morfosintáctico, sintáctico, semántico y/o un análisis del discurso de la sentencia.
- Identificar la frase que responda a la pregunta de la sentencia.
- Reemplazar la frase con un tipo de pregunta (wh-word) adecuado.
- Hacer que el sujeto esté en concordancia con el verbo e invertirlos.
- Post-procesado de la pregunta, que genera una bien formada pregunta.

Los que este artículo propone para el francés, tienen dos diferencias con respecto al inglés.

- Invertir sujeto-verbo.
- Modificaciones en la forma verbal, debido a las modificaciones sintácticas.

Un aspecto importante del cual nos informan tiene que ver con la tipología de las preguntas, estas son generadas en base a una transformación sintáctica aplicada al texto de origen (texto a ser evaluado), para ello se identifican qué componentes sintácticos son la mejor elección para la generación de la pregunta, los cuales se basan en la intuición y en estudios lingüísticos.

Después se clasifican estos componentes basados en su función gramatical. En la siguiente tabla se puede ver a qué nos referimos:

Function	Example	Constituent
Subject	S: Terminer ses études est l'objectif de John. *To complete his studies is John's goal.* Q: Quel est l'objectif de John ? *What is John's goal?*	Noun Phrase Pronoun Infinitive clause That/What clause
Direct object	S: J'ai vu le père de Marie. *I saw Mary's father.* Q: Qui as-tu vu ? *Whom did you see?*	Noun phrase Pronoun Subordinate clause
Indirect object	S: J'ai offert ce cadeau à mon frère. *I offered this present to my brother.* Q: À qui as-tu offert ce cadeau ? *To whom did you offer this present?*	Noun phrase Pronoun
Subject complement	S: La peinture est très belle. *The painting is beautiful.* Q: Comment est la peinture ? *What is the painting?*	Noun phrase Adjective
Locative adverbial adjunct	Q: Je viens du cinéma. *I come from the movie theater.* Q: D'où viens-tu ? *Where you do come from?*	Noun phrase Pronoun
Temporal adverbial adjunct	S: Mon avion décolle à 13h. *My plane takes off at 1 pm.* Q: À quelle heure décolle ton avion ? *When does your plane take off?*	Noun phrase Pronoun
Appositive	S: Le président français, Nicolas Sarkozy, ... *The French president, Nicolas Sarkozy, ...* Q: Qui est Nicolas Sarkozy ? *Who is Nicolas Sarkozy?*	Noun phrase
Parenthesised acronym	S: L'Organisation des Nations unies (ONU) ... *The United Nations (UN) ...* Q: Que signifie ONU ? *What does UN mean?*	Noun phrase

Tabla 3-1: Tipología de los tipos de preguntas y sus funciones gramaticales.

Para el análisis de las sentencias se usan dos analizadores sintácticos diferentes, de los cuales cabe destacar XIP.

- XIP (Xerox Incremental Parsing) Combina el análisis sintáctico con las entidades y produce un árbol constituyente superficial.

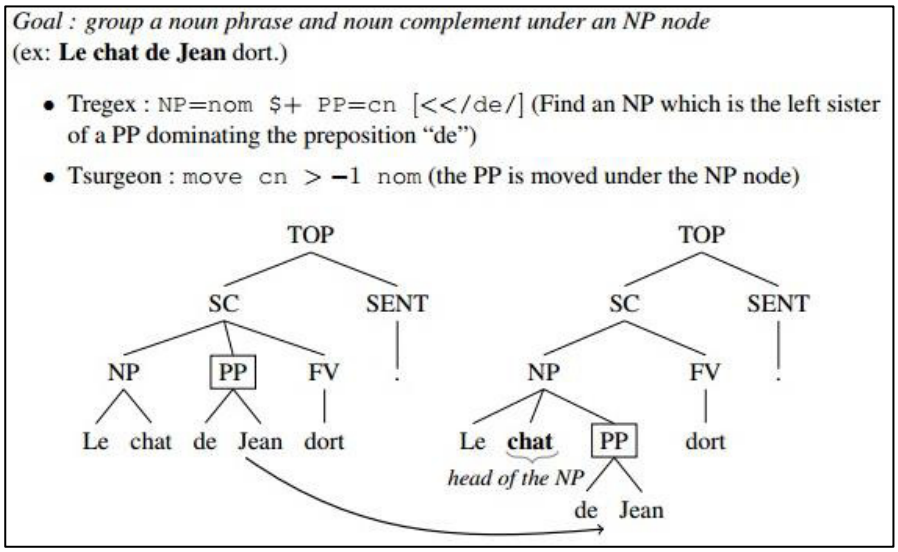


Figura 3-1: Ejemplo en forma de árbol del analizador XIM.

Para el proceso de evaluación tuvo como objetivo responder las siguientes preguntas:

- ¿Cuál es la calidad de las preguntas generadas? ¿Qué cantidad de post-edición es necesaria para producir preguntas perfectas?
- ¿Cuál es la llamada del sistema?
- ¿Qué tan bien se genera una pregunta que tenga en la frase una substantivación?
- ¿Cuál es el desempeño una vez analizada las preguntas automáticas para la búsqueda de respuestas?

3.3. **G-Asks: An intelligent Automatic Question Generation System for Academic Writing Support (8)**

El artículo presenta un novedoso generador de preguntas llamado G-Asks, el cual genera unas preguntas específicas como una forma de apoyo para los estudiantes a través de la escritura.

Hicieron un estudio a una gran escala en la cual participaron 24 personas encargadas de la supervisión y 33 estudiantes de investigación y compararon las preguntas generadas por G-Asks con preguntas generadas por personas.

Como conclusión se obtuvo que G-Asks genera preguntas tan útiles como las formuladas por las personas encargadas de la supervisión.

Un aspecto importante a destacar es la taxonomía de las preguntas las cuales se definieron siguiendo 18 categorías de preguntas, estas categorías están sacadas de Graesser, Person, and Huber (1992),

Categoría de la pregunta	Especificación abstracta
1. Verificación	Invita a dar una respuesta afirmativa o negativa.
2. Disyuntivo	¿Es X, Y o Z?
3. Conceptual	¿Quién?, ¿Qué?, ¿Dónde?, ¿Cuándo?
4. Ejemplar	¿Un ejemplo de X?
5. Especificativa	¿Cuáles son las propiedades de X?
6. Cuantificación	¿Cuánto?, ¿Cuántos?
7. Definición	¿Qué significa X?
8. Comparación	¿Qué tan similares son X e Y?
9. Interpretación	¿Qué significa la X?
10. Antecedente causal	¿Por qué / Como ocurrió X?
11. Consecuencia causal	¿Y ahora qué?, ¿Y si?
12. Orientativa	¿Por qué un agente hace X?
13. Instrumental / procedimental	¿Cómo un agente hace X?
14. Habilitación	¿Qué ocurre si se activa X?
15. Expectación	¿Por qué no ocurre X?

16. Sentencioso	¿Qué opinas de X?	
17. Asertivo		
18. Solicitud / Directiva		

Tabla 3-2: Categorías de preguntas de Grasser, Person, and Huber (1992).

En su estudio el cual tuvo alrededor de 8000 preguntas se encontraron con frecuencia cuatro categorías: verificación, Instrumental / procedimental, conceptual, y cuantificación.

3.4. **Dos aproximaciones para generar preguntas en euskera (9)**

El artículo presenta por un lado un generador de preguntas basado en chunks (10), por otro lado un generador de preguntas basado en dependencias sintácticas (11).

Estos generadores están basados en:

- Generación de preguntas basado en el sintagma nominal.
- Características semánticas para la generación de preguntas.

Como resultado se obtienen dos generadores de preguntas:

- QG-Malti: basado en Maltixa, un programa de análisis de dependencias para el euskera.
- QG-Ixati: basado en Ixati, un analizador sintáctico para el euskera, el cual identifica los elementos constituyentes de la frase.

Para la evaluación se eligieron textos relacionados con la ciencia y la tecnología para alumnos de secundaria, ya que una de las metas está ligada a la enseñanza.

Una de las prioridades fue focalizar la evaluación en la sintaxis y la fluidez de las preguntas generadas. Para ello se contó con una persona que tendría el rol de juez y seguiría el mismo criterio de clasificación propuesto en (12). También se hizo un estudio sobre la calidad del tipo de preguntas determinando si los wh-words generados tenían concordancia con la frase de origen.

3.5. **Generación de Preguntas a partir de Mapas Conceptuales (13)**

Este artículo presenta un enfoque de generación de preguntas para un texto que contiene diálogos. Está basado en unas teorías psicológicas que plantean la hipótesis de que las preguntas son generadas a partir de una representación de conocimiento modelado como un mapa conceptual. El modelo extrae mapas conceptuales de un libro de texto y estos son usados para generar preguntas.

Un ejemplo básico de cómo sería un mapa conceptual:

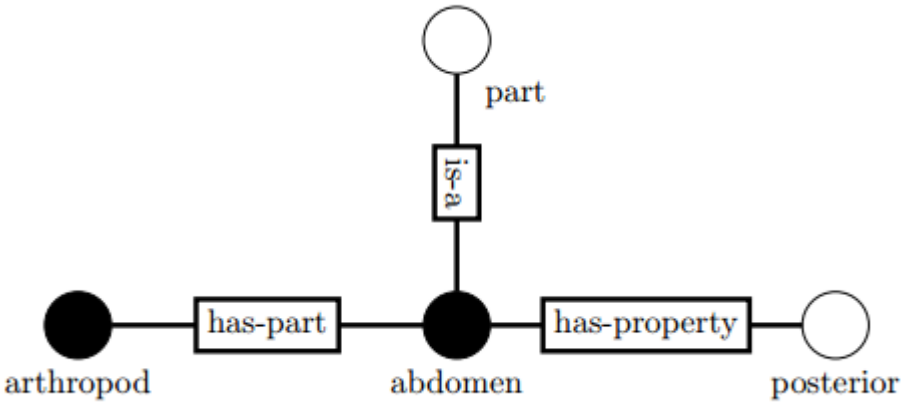


Figura 3-2: Fragmento de un mapa conceptual.

El mapa conceptual generado en este artículo viene dado por dos estructuras elementales que son:

- Términos clave: Son representados mediante un nodo negro de los cuales se puede formar un triple, esto quiere decir que puede salir del nodo negro tres aristas que acaben en otros nodos. En la figura 3-2 podemos observar como del abdomen salen tres aristas.

- Aristas etiquetadas: Es un pequeño conjunto de aristas que puede representar un gran porcentaje de las relaciones en un domino.

En términos generales el propósito de este estudio es generar y evaluar preguntas pedagógicamente apropiadas a diferentes niveles de especificidad a través de una o más frases.

Capítulo 4

Herramientas

Antes de poder elaborar cualquier proyecto de esta índole es necesario adquirir las herramientas que nos ayudarán a desarrollarlo, en este capítulo nos encargaremos de nombrar y explicar detalladamente todas las herramientas que se han usado.

4.1. IXA pipes (14)

Es un conjunto de herramientas para el Procesamiento de Lenguaje Natural. Dependiendo de la herramienta usada esta puede estar disponible en distintos idiomas tales como el castellano, euskera, italiano, inglés y/o gallego.

Estas herramientas han sido desarrolladas por el grupo IXA NLP de la Universidad del País Vasco.

Para trabajar con estas herramientas se puede acceder desde las siguientes webs:

- Maven Central Repository : http://search.maven.org/
- Ixa Pipe Tools: http:// ixa2.si.ehu.es/ixa-pipes/

Hay distintas formas de adherir las librerías al proyecto. En el apéndice 1 se muestra como incorporarlas a la plataforma Eclipse, que es la que usamos para desarrollar el generador de preguntas.

De este conjunto de herramientas cabe destacar las siguientes cinco, ya que han sido piezas claves en la generación de preguntas.

Herramienta	Función
Ixa-pipe-tok	Segmentación y tokenización
Ixa-pipe-pos	Etiquetado gramatical
Ixa-pipe-nerc	Etiquetado para el reconocimiento de entidades
Ixa-pipe-parse	Analizador morfo-sintáctico
Ixa-pipe-srl	Analizador sintáctico basado en dependencias.

Tabla 4-1: Herramientas de Ixa pipe.

Hay que destacar que las herramientas tienen más funciones de las explicadas en los puntos de abajo, pero en este proyecto solo se usan las que se mencionan.

4.1.1. Ixa-pipe-tok

Esta herramienta nos permite tokenizar y separar por sentencias una entrada que esté formada por un texto plano.

Es necesario que el texto este completamente tokenizado y segmentado, para que el resto de las herramientas funcionen.

A continuación se muestra los pasos realizados con capturas de código para poder usar la herramienta.

- Creamos las siguientes variables:

```
BufferedReader breader;
TokenFactory tokenFactory;
Properties properties;
Tokenizer<Token> tokenizer;
Segmenter segmenter ;
List<Token> listaTokens;
List<List<Token>> sentences;
```

  - BufferReader: Esta clase es la que nos permitirá leer un fichero de texto.
  - TokenFactory: Esta clase provee la funcionalidad de crear objetos Token.
  - Properties: La clase Properties representa un conjunto de propiedades.
  - Tokenizer: Esta clase separa el texto en objetos individuales
  - Segmenter: Esta clase divide el texto tokenizado en sentencias.

- Elegimos el fichero a tokenizar y lo leemos mediante la clase BufferedReader

```
String dir = "arhivo.txt";
breader =  new BufferedReader(new FileReader(dir));
```

- Inicializamos el tokenFactory

```
tokenFactory = new TokenFactory();
```

- Inicializamos las properties(propiedades)

```
properties = new Properties();
```

- Inicializamos el tokenizer pasándole como parámetros breader, tokenFactory y properties.

```
tokenizer = new AncoraTokenizer<Token>(breader,
tokenFactory, properties);
```

- Inicializamos el segmenter

```
segmenter = new Segmenter();
```

- Inicializamos la listaTokens (lista de palabras) y mediante una función del tokenizer le pasamos todas las palabras tokenizadas en forma de lista

```
listaTokens = new ArrayList<Token>();
listaTokens = tokenizer.tokenize();
```

- Obtenemos las sentencias a partir de la lista de palabras

```
sentences = segmenter.segment(listaTokens);
```

Una vez realizado todos estos pasos, en la variable “sentences”, que es una lista en la que cada índice está formado por otra lista, se agruparán todas las sentencias encontradas pertenecientes a un párrafo. De esta forma en el índice 1 de la variable “sentences”, podrá haber X sentencias pertenecientes al primer párrafo.

4.1.2. lxa-pipe-pos

Esta herramienta realiza un etiquetado gramatical, basándose en las etiquetas EAGLES, el cual es el proceso de asignar a cada una de las palabras de un texto un código el cual nos indicará a que categoría pertenece, cuáles son sus atributos y valores que puede tomar.

Las etiquetas Eagles son un conjunto de etiquetas basadas en las etiquetas propuestas por el grupo EAGLES para la anotación morfosintáctica de lexicones y corpus para todas las lenguas europeas.

Para cada categoría se presentan los atributos, valores y códigos que puede tomar.

Estas categorías son:

- Adjetivos, adverbios, artículos, determinantes, nombre, verbos, pronombres, conjunciones, numerales, interjecciones, abreviaturas, preposiciones y signos de puntuación.

Ejemplo:

La palabra “el” tiene la siguiente etiqueta formado por distintos códigos: TDMS0

El siguiente cuadro muestra la relación entre el código, el atributo y el valor.

ARTÍCULOS			
Pos.	**Atributo**	**Valor**	**Código**
1	Categoría	Artículo	T
2	Tipo	Definido	D
3	Género	Masculino	M
		Femenino	F
		Común	C
4	Número	Singular	S
		Plural	P
5	Caso	-	0

Tabla 4-2: Tabla de la categoría Artículos.

De esta forma sabemos que “el” es un Artículo de tipo definido, género masculino, y número singular.

Para cada una de las categorías hay una tabla que nos da información sobre qué significa el código (15).

Esta herramienta también usa un modelo externo que es de donde extrae la información lingüística, concretamente para el español existen dos modelos basados en el corpus de AnCora (16).

A continuación se muestran los pasos realizados con capturas de código para poder usar la herramienta.

- Creamos las siguientes variables

```
MorphoFactory morphoFactory;
MorphoTagger postTagger;
List<Morpheme> Morfemas;
Properties propertiesPos;
```

  - MorphoFactory: Clase para la creación de morfemas.
  - MorphoTagger: Clase que mediante un modelo se encargará de etiquetar todas las palabras de una sentencia.
  - List<Morpheme>: Clase que genera una lista de morfemas.

- En las propiedades se especificará que el lenguaje será español y se elegirá el modelo a cargar.

```
propertiesPos.setProperty("language", "es");
propertiesPos.setProperty("model", "modelo_ancora.bin'
```

- Inicializamos el morphoFactory y a continuación inicializamos el postTagger pasándole como atributos las propiedades donde se encuentra el modelo, y el morphoFactory.

```
morphoFactory = new MorphoFactory();
postTagger = new MorphoTagger(propertiesPos
, morphoFactory );
```

- Se deberá convertir las sentencias pertenecientes a la lista “sentences” a una lista de Strings y esta lista de Strings de una sentencia es la que se pasará al postTagger, que mediante la función “getMorphemes” devolverá una lista de morfemas.

```
String[] sentencia = sentToString(sentences.get(0));
morfemas = postTagger.getMorphemes(sentencia);
```

- Esta lista de morfemas tiene una función que permite ver el código asignado de cada una de las palabras de la sentencia.

```
morfemas.get(i).getTag();
```

4.1.3. lxa-pipe-nerc

Esta herramienta etiqueta entidades gramaticales que reconoce en un texto. Hay 4 tipos de entidades que reconocen, localización, persona, organización, y Misc.

Al igual que la herramienta lxa-pipe-pos, también se apoya en un modelo.

Ejemplo:

En la siguiente frase, "*Francisco luchó en la batalla de Roma.*"

La herramienta es capaz de devolver las siguientes entidades una vez que se analice la frase:

- Francisco - persona
- Roma - localización

A continuación se muestra los pasos realizados con capturas de código para poder usar la herramienta.

- Creamos las siguientes variables

```
Properties annotateProperties
NameFactory nameFactory;
StatisticalNameFinder nameFinder;
```

  - NameFactory: Esta clase provee la funcionalidad para crear objetos del tipo Name.
  - StatitisticalNameFinder: Esta clase es la que mediante un modelo externo crea una lista e identifica las entidades.

- Inicializamos las propiedades, indicando el idioma y donde se encuentra el modelo

```
annotateProperties = new Properties();
annotateProperties.setProperty("language", "es");
annotateProperties.setProperty("model", "modNerc.bin")
```

- Inicializamos el nameFactory y a continuación el nameFinder pasándole como parámetros las propiedades y el nameFactory.

```
nameFactory = new NameFactory();
nameFinder = new StatisticalNameFinder(annotatePropert
, nameFactory);
```

- Se deberá convertir las sentencias pertenecientes a la lista "sentences" a una lista de Strings y esta lista de Strings se pasarán a una función del nameFinder que devolverá una lista perteneciente a la clase Span (en este contexto se usa esta clase para saber la posición de inicio y de final de una sentencia en la cual se ha encontrado una entidad).

```
String[] sentencia = sentToString(sentences.get(0));
Span[] valor = nameFinder.nercToSpans(sentencia);
```

4.1.4. lxa-pipe-parse

Esta herramienta realiza un análisis morfosintáctico de un texto, para ello se apoya en un modelo lingüístico que provee toda la información para el análisis.

En el capítulo 2.2 se explica en qué consiste un análisis morfosintáctico.

En esta herramienta la parte relacionada con el análisis sintáctico de analizador morfo-sintáctico obvia algunas funciones sintácticas como el sujeto, predicado, complemento circunstancial, etc...

A continuación se muestra los pasos realizados con capturas de código para poder usar la herramienta.

- Creamos las siguientes variables

```
ConstituentParsing parse;
Properties annotateProperties;
Parse par;
```

  - ContituentParsing: Esta es la clase donde se carga el modelo.
  - Parse: Esta clase es la encargada de realizar el análisis morfológico.

- Iniciamos las propiedades e identificamos la ruta del modelo.

```
annotateProperties = new Properties();
annotateProperties.setProperty("model", "modeParse.bin
```

- Iniciamos el parse, pasándole las propiedades como parámetro al constructor.

```
parse = new ConstituentParsing(annotateProperties);
```

- Se deberá convertir las sentencias pertenecientes a la lista “sentences” a una lista de Strings y esta lista de Strings se pasará a una función del parse que se encargará de devolver la sentencia analizada.

```
String sentencia = sentToString(sentences.get(0));
par = parse.parse(sentencia, 1);
par.show();
```

- Lo que la salida del sistema mostrará será lo siguiente:

```
(TOP (SENTENCE (SN (DAOFSO La) (GRUP.NOM (NCFSOOO familia)....) ) )
```

La variable “par” tiene distintas funciones, una de ellas es la función “getChildren()”, esta función lo que hará será devolver un array del tipo Parse, el cual se encontrará a un nivel mayor de profundidad.

- Ejemplo de las funciones de la variable “par”:

  La batalla acabó.

  (SENTENCE (SN (DA0FS0 la) (GRUP.NOM (NCFS000 batalla))) (GRUP.VERB (VMIS3S0 acabó)) (FP .))

En un principio la variable “par” nos situará al nivel de “SENTENCE”.

  - Mediante la función “getType()” de estar variable “par” podremos obtener el tipo, que en este caso será “SENTENCE”.
  - Mediante la función “getCoveredText()” se podrá obtener el texto el cual estará formado por todo el contenido que hay entre los paréntesis verdes a excepción del propio tipo.
  - Mediante la función “getSize()” se podrá obtener el número de hijos que tiene el padre, en este caso concreto sería de tres.
  - Al usar la función “getChildren()” en la variable “par”, este devolverá un array del tipo Parse. Cada índice de este nuevo array estará formado por los hijos de “SENTENCE”, estos son:

    Índice 0 (SN (DA0FS0 la) (GRUP.NOM (NCFS000 batalla)))

    Índice 1 (GRUP.VERB (VMIS3S0 acabó))

    Índice 2 (FP .)

Llegados a este punto se puede observar la forma de árbol que esta tiene, si se siguiese desglosando podríamos observar que iría tomando esta forma. En la figura 2-2 del apartado 2.2 se muestra cómo quedaría gráficamente.

4.1.5. Ixa-pipe-srl

Esta herramienta sirve como analizador de los roles semánticos y como analizador basado en dependencias sintácticas. Es la segunda opción la que se ha usado en este proyecto, en el capítulo 2.3 se explica en qué consiste un análisis basado en dependencias.

Una de las novedades usadas con respecto a las anteriores herramientas es el formato NAF. El formato NAF (NLP Annotation Format), es un formato de anotación lingüística diseñada para las tuberías (pipelines) del PLN. Las herramientas pueden usar la librería kaflib para una sencilla integración del formato NAF. En otras palabras se trata de una librería que provee una estructura generalizada que puede almacenar distintas anotaciones que usen las herramientas del PLN. De esta forma dentro de este formato se pueden guardar distintos resultados de distintos análisis, nos ahorra tener que crear diversas estructuras de datos para almacenar los resultados que cada análisis realiza.

El formato NAF no es algo exclusivo de esta herramienta, el resto de las herramientas también las pueden utilizar, pero no es hasta el uso de esta herramienta, o mejor dicho, no es hasta el momento de realizar un análisis basado en dependencias que se empieza a usar, esto ocurre por la sencilla razón de que el formato obligatorio de entrada de la herramienta ixa-pipe-srl viene dado en formato NAF, en el resto de herramientas los formatos de entrada pueden ser o NAF o un texto plano.

A continuación se muestra los pasos realizados con capturas de código para poder usar la herramienta.

- Creamos las siguientes variables y las inicializamos.

```
CargarKaf kafNuevo = new CargarKaf("es","1.1.12");
CargarToken token = new CargarToken("src/texto/doc1.t
CargarPOS pos = new CargarPOS();
```

  - CargarKaf: esta clase genera un documento NAF (mediante la librería KAF), al cual le pasamos el idioma que queremos en que se genere y la versión.
  - CargarToken: esta clase carga la herramienta Ixa-pipe-tok vista en el capítulo 4.1.1, pasándole directamente el texto que deseamos tokenizar.
  - CargarPOS: esta clase carga la herramienta Ixa-pipe.pos vista en el capítulo 4.1.2.

- Una vez tengamos tokenizado el texto, este resultado se deberá guardar en el documento NAF, de igual manera usando el ixa-pipe-pos, se realizará un etiquetado gramatical el cual se guardará también en el documento NAF.

```
token.getAnnotate().tokenizeToKAF(kafNuevo.getKaf());
pos.getAnnotate().annotatePOSToKAF(kafNuevo.getKaf())
```

- Hasta este punto hemos usado herramientas anteriores que son necesarias para el funcionamiento de la actual, a continuación creamos la siguiente variable perteneciente a la clase CargarSRL y la inicializamos de la siguiente forma:

```
CargarSRL srl = new CargarSRL("spa","only-deps");
```

  - CargarSRL: esta clase carga el analizador basado en dependencias, sus dos parámetros indicarán que estará en castellano y que solo usaremos las dependencias.

- Una vez tengamos inicializado la variable srl, se deberá guardar este resultado en el documento NAF.

```
srl.getAnnotate().SRLToKAF(kafNuevo.getKaf(), "spa","o
```

- Por último, para obtener la lista de dependencias se deberá hacer lo siguiente:

```
List<Dep> listaDep =  kafNuevo.getKaf().getDeps();
```

En la figura 2-3 del apartado 2.3 se puede ver un resultado orientativo.

Capítulo 5

El generador de Preguntas - Desarrollo

El proyecto tiene como objetivo principal la generación de preguntas a nivel de sentencia a partir de una entrada de un texto plano por medio de un análisis morfo-sintáctico y por medio de un análisis basado en dependencias sintácticas.

El primer paso para la creación de un generador de preguntas es definir el alcance, saber sobre qué tipo de textos se va a trabajar. A la hora de definir el alcance, lo que estamos definiendo también es la algoritmia que se usará para generar las preguntas. Cuando se realiza un generador de preguntas sobre un texto de historia las preguntas que se buscan son del tipo histórico, y estas están estrechamente relacionadas con personajes, lugares y fechas, con situaciones causales, ¿Cuándo sucedió?, ¿Dónde?, ¿Quién?, ¿Quiénes?...

En este proyecto, como ya se comentó en el alcance, los textos elegidos son del tipo histórico enfocado al tema de guerras. Se decidió esta temática por dos razones fundamentales. La primera es que la vida y duración del proyecto no permite enfocarse en distintas temáticas. La segunda es porque se consiguió encontrar textos históricos centrados en las guerras, las cuales valdrían para ser el Corpus y de esta manera tener diversos textos para el training y para la evaluación.

A continuación se presenta un ejemplo de una sentencia extraída de un texto histórico enfocado en la guerra y los tipos de preguntas que se deberían obtener a partir de ella.

La batalla de Argentovaria tuvo lugar en mayo del año 378, en Argentovaria.

Preguntas generadas:

1. *¿Qué tuvo lugar en mayo del año 378, en Argentovaria?*
2. *¿Cuándo tuvo lugar la batalla de Argentovaria?*
3. *¿Dónde tuvo lugar en mayo del año 378 la batalla de Argentovaria?*

La arquitectura o las etapas por las que se pasan para poder generar las preguntas son las mismas en los casos de realizarlas por medio de un analizador morfosintáctico o un analizador basado en dependencias sintácticas, en ambos casos se usa la estructura propuesta por Rus and Graesser (17) la cual consiste en:

1. Selección del objetivo.
2. Selección del tipo de pregunta.
3. Construcción de la pregunta.

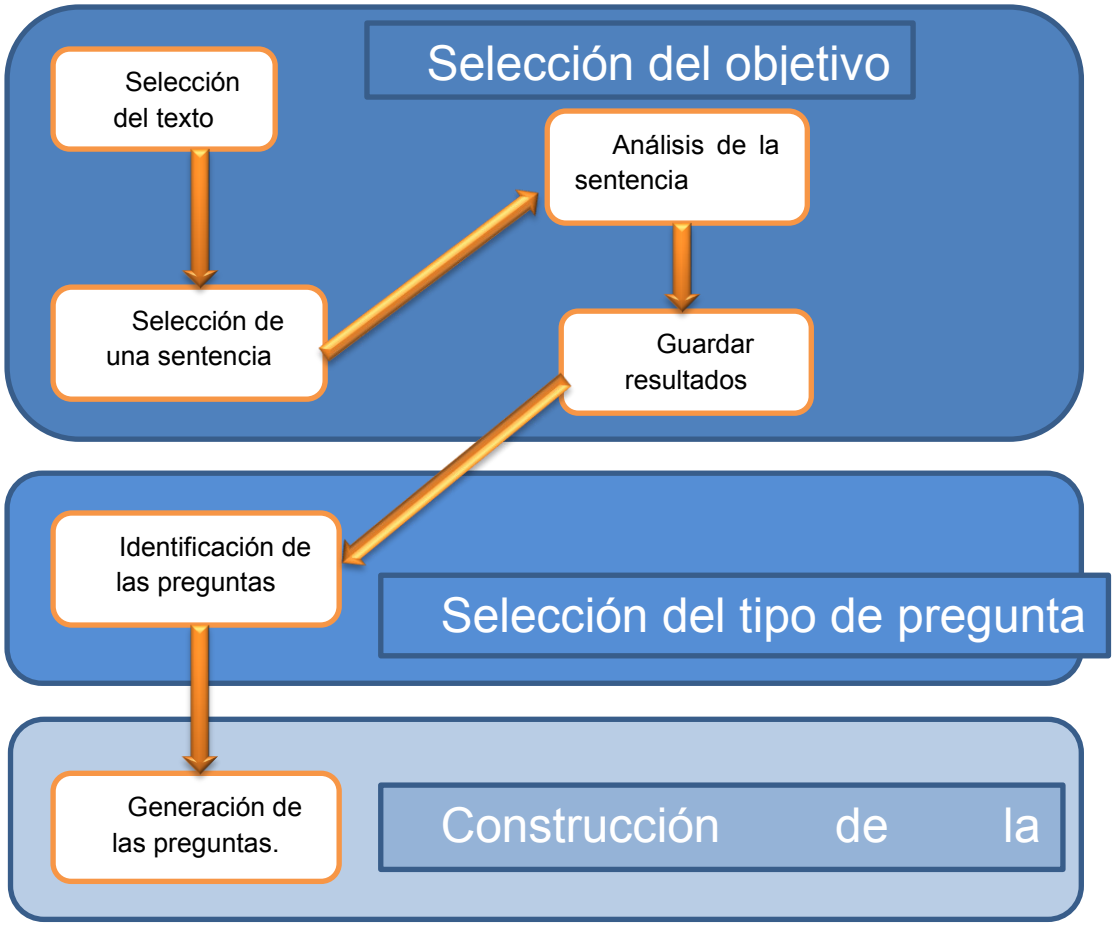


Figura 5-1: Arquitectura del generador de preguntas.

El contenido de este capítulo consta de los siguientes apartados:

- En el apartado 5.1 nos centraremos en hablar sobre el corpus, la obtención, el pre-procesamiento, las modificaciones y soluciones.
- En el apartado 5.2 se profundiza en la arquitectura, dando detalles sobre cómo es su funcionamiento.
- En el apartado 5.3 y 5.4 se hablaran de los programas desarrollados en base a los dos analizadores usados.

5.1. El corpus

El corpus, o el conjunto de documentos textuales, de este proyecto ha sido extraído en su totalidad de la página web Wikipedia. En un primer momento se pensó en utilizar textos relacionados con la temática de historia enfocada a alumnos de un nivel secundario, ya que se pensó que la lingüística usada en los textos sería sencilla, pero el tamaño del corpus encontrado no fue el suficiente para poder formar un corpus. Por esta razón se recurrió a la búsqueda de textos de historia con una única condición, que estas estuviesen centradas en el tema de guerra.

Y ¿por qué el tema de guerras?, en un primer momento se pensó que limitando más la temática se conseguiría mayor precisión a la hora de generar la pregunta.

Uno de los anexos de historia de la web Wikipedia es el tema de las guerras. En las cuales podemos encontrar diversos textos de guerras organizados por siglos.

AÑO	BATALLA	RESUMEN
709	Batalla de Kabul	Conquista islámica de Afganistán.
711 (19 de julio)	Batalla de Guadalete	El ejército visigodo es derrotado por el ejército musulmán.
717 (15 de agosto)	Sitio de Constantinopla	Constantinopla resistió durante un año el sitio de los árabes.
721 (9 de junio)	Batalla de Toulouse	El ejército musulmán es derrotado por Odón de Aquitania.
722	Batalla de Covadonga	Las tropas de Don Pelayo derrotan por primera vez a los musulmanes que habían invadido la Península Ibérica.
723	Batalla de Balanjar	Las tropas del Janato Jázaro son derrotados por el ejército del Califato Omeya.
732 (11 de octubre)	Batalla de Poitiers (Tours)	Las tropas moras son derrotadas por Carlos Martel cerca de Poitiers, impidiendo así su acceso al resto de Europa.
734	Batalla del Boarn	Carlos Martel conquista el oeste de Frisia.
746	Batalla de Rupar Thutha	Kufa y Mosul son capturados por Marwan II.

Tabla 5-1: Batallas del siglo VIII

Los textos por lo general tenían esta forma:

Batalla de Tolosa (721)

La **batalla de Tolosa** (721) fue una victoria del ejército franco comandado por el duque Odón de Aquitania, conocido también como Eudes de Aquitania, sobre el ejército musulmán del Califato Omeya.

As-Samh ibn Malik al-Jawlani, gobernador (o valí) musulmán hispano, forjó un poderoso ejército desde África del Norte, Yemen y Siria para conquistar Aquitania, un gran ducado al suroeste de la actual Francia, formalmente bajo soberanía de los francos, pero en la práctica, casi independiente a manos de los duques de Aquitania.

Al-Jawlani puso asedió a la ciudad de Tolosa, por entonces la ciudad aquitana más importante. El duque Eudes de Aquitania se vio forzado a partir para solicitar ayuda, regresando tres meses más tarde, justo cuando la ciudad estaba a punto de rendirse y caer en manos de los musulmanes invasores el 9 de junio de 721, en lo que ahora se conoce como la *Batalla de Tolosa*. La victoria en esta batalla consiguió temporalmente evitar el avance del control musulmán más hacia el oeste de Narbona, hacia tierras aquitanas.

Figura 5-2: La batalla de Tolosa

Estos textos eran directamente guardados en formato txt como parte del corpus, con el nombre del título.

En algunos casos el texto extraído se descomponía en distintas categorías. Estas categorías podían ser: antecedentes, la batalla, y conclusión. En estos casos se guardaba todo el texto pero se borraban los títulos de las categorías, ya que lo que nos interesa es tener un único texto formado simplemente por párrafos.

De Wikipedia se extrajo un total de 300 textos de los cuales 210 servirían para realizar el training (70 % de los textos) en estos textos hay 3509 sentencias con un total de 99499 palabras, el resto (30% de los textos) se reservó para la evaluación.

5.1.1. Pre-procesamiento del corpus.

Una vez obtenido el corpus se hizo unas modificaciones sobre estos, estas eran de carácter obligatorio, ya que no se podía realizar el análisis morfo-sintáctico a partir de ellas.

Referencias numéricas

En algunas ocasiones al obtener un texto este venía con referencias numéricas en algún lugar del texto, a continuación se muestra como se veían estas referencias antes y después de ser pasadas a un fichero txt.

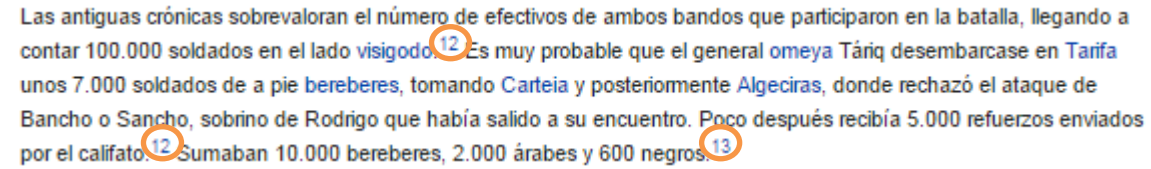

Las antiguas crónicas sobrevaloran el número de efectivos de ambos bandos que participaron en la batalla, llegando a contar 100.000 soldados en el lado visigodo.[12] Es muy probable que el general omeya Tárik desembarcase en Tarifa unos 7.000 soldados de a pie bereberes, tomando Carteia y posteriormente Algeciras, donde rechazó el ataque de Bancho o Sancho, sobrino de Rodrigo que había salido a su encuentro. Poco después recibía 5.000 refuerzos enviados por el califato.[12] Sumaban 10.000 bereberes, 2.000 árabes y 600 negros.[13]

Figura 5-3: Texto con referencias numéricas de Wikipedia.

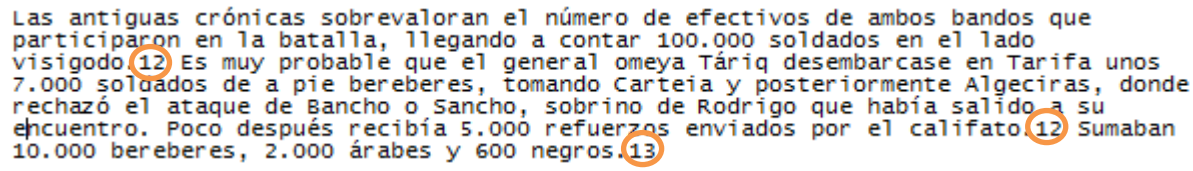

Las antiguas crónicas sobrevaloran el número de efectivos de ambos bandos que participaron en la batalla, llegando a contar 100.000 soldados en el lado visigodo.12 Es muy probable que el general omeya Tárik desembarcase en Tarifa unos 7.000 soldados de a pie bereberes, tomando Carteia y posteriormente Algeciras, donde rechazó el ataque de Bancho o Sancho, sobrino de Rodrigo que había salido a su encuentro. Poco después recibía 5.000 refuerzos enviados por el califato.12 Sumaban 10.000 bereberes, 2.000 árabes y 600 negros.13

Figura 5-4: Texto con referencias numéricas en formato txt.

Estas referencias numéricas no forman parte del texto narrativo y por lo tanto no pueden formar parte del análisis morfo-sintáctico.

Para solucionar este inconveniente se realizaron los siguientes pasos:

- Se estudiaron aspectos generales que compartían los textos que tenían referencias numéricas. Estas siempre venían al final de una sentencia, después del punto. En algunas ocasiones habían varias referencias seguidas.
- Se pasó el programa por todo el corpus y se devolvió un nuevo corpus sin estas referencias numéricas.

Signos de puntuación

Otro de los problemas fue con respecto a los signos de puntuación:

- Paréntesis: Dentro de estos se encontró en una gran mayoría de palabras en otros idiomas, las cuales serían un problema a la hora de pasarlo por el analizador morfo-sintáctico en español.

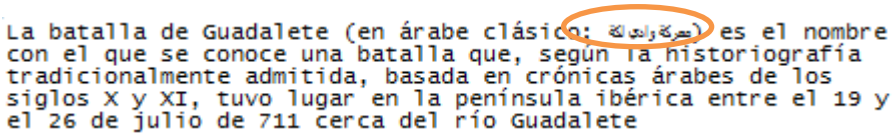

La batalla de Guadalete (en árabe clásico: معركة وادي لكة) es el nombre con el que se conoce una batalla que, según la historiografía tradicionalmente admitida, basada en crónicas árabes de los siglos X y XI, tuvo lugar en la península ibérica entre el 19 y el 26 de julio de 711 cerca del río Guadalete

Figura 5-5: Texto con un fragmento en árabe.

Solución

Eliminar todos los paréntesis del corpus.

- Punto (.): Concretamente el punto seguido, el problema viene en el momento que se encuentra un texto en el cual hay un punto seguido sin dejar espacio entre este y la palabra que viene a continuación. El analizador morfo-sintáctico detectaba como una misma sentencia tanto a la sentencia de la izquierda como al de la derecha.

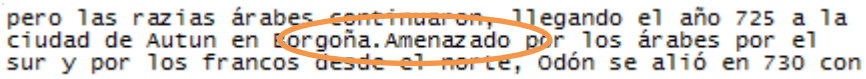


Figura 5-6: Texto con punto seguido.

Solución
Cada vez que se encuentre un punto seguido dejar un espacio en blanco entre este y la siguiente palabra.

- Guion largo (-): El problema viene dado por culpa de los nombres de lugares o personajes históricos, en algunas ocasiones algunos nombres estaban compuestos por medio de un guion largo. Los nombres que tenían un guion largo al ser pasados por el analizador morfo-sintáctico no eran considerados como una misma entidad.

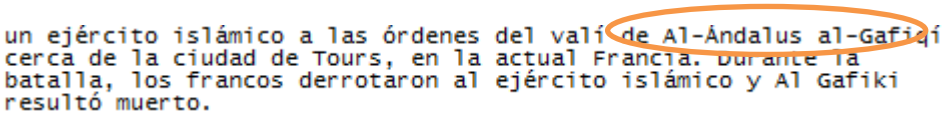


Figura 5-7: Texto con guion largo.

Solución
Eliminar los guiones largos, ya que nos es trivial que una entidad tenga un guion, solo nos interesa que la palabra sea reconocida como tal.

El programa desarrollado para tratar el pre-procesamiento del corpus se encuentra en el Apéndice 1

- Código del pre-procesamiento del Corpus.
  - Main

5.2. **Arquitectura**

La arquitectura está formada por estas tres etapas:

5.2.1. Selección del objetivo.

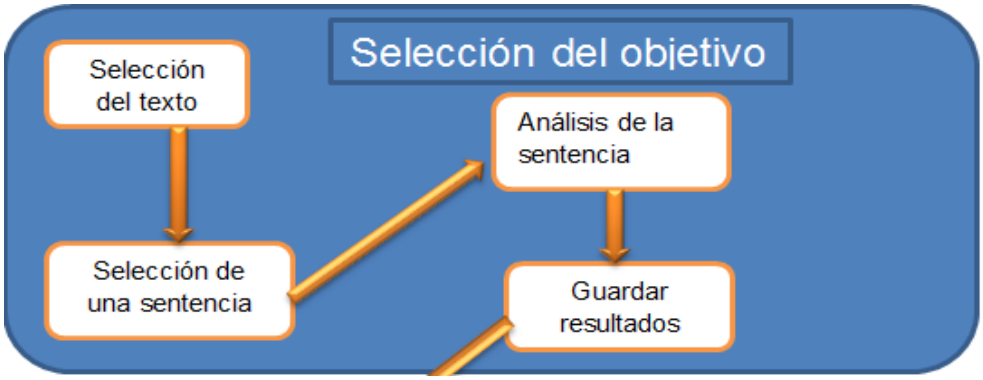


Figura 5-8: Selección del objetivo

En esta etapa se decide cual será la frase a partir de la cual se aplicarán diversos algoritmos para determinar qué tipos de preguntas se pueden formular.

Selección del texto

En un primer momento el usuario deberá seleccionar un texto con el cual se deberá trabajar, como ya se ha explicado hay un total de 210 textos que serán usados para el training.

Selección de una sentencia

Una vez que se haya seleccionado uno de los 210 textos, dependiendo con qué generador se esté trabajando la selección será de la siguiente manera:

- Generador a partir de un analizador morfosintáctico, la selección de la sentencia será de manera automática. El sistema se encargará de ir seleccionando una a una todas las sentencias.
- Generador a partir de un analizador basado en dependencias, la selección de la sentencia será de manera manual. El usuario será el que se encargue de seleccionar una de todas las sentencias del texto.

En el generador de preguntas a partir de un analizador morfosintáctico se analizan todas las sentencias ya que el tiempo medio de análisis de **todo un texto** es de 4 segundos, mientras que en el otro generador el tiempo medio de análisis **de cada sentencia** es de 10 segundos.

Análisis de la sentencia

Después de obtener la sentencia, se procederá a realizar un análisis. En nuestro caso, como ya se ha comentado, los análisis podrán ser mediante dos tipos:

- Análisis morfosintáctico.
- Análisis basado en dependencias sintácticas.

Ambos análisis son independientes el uno del otro, no están vinculados de ninguna manera. Son programas que se ejecutan al margen uno del otro, el procedimiento y la arquitectura de ambos es la misma, a excepción del análisis.

Guardar resultados

Por último se deberá guardar el resultado de los análisis, serán un apoyo fundamental a la hora de seleccionar el tipo de preguntas sobre las sentencias.

5.2.2. Selección del tipo de pregunta.

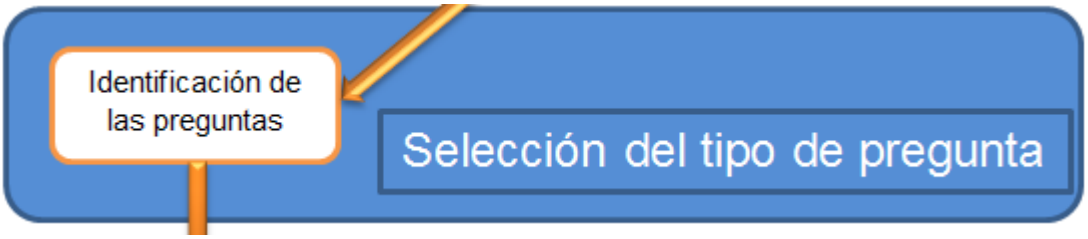


Figura 5-9: Selección del tipo de pregunta

En esta etapa en base a los resultados guardados en la etapa anterior se procederá a identificar qué tipos de preguntas se pueden generar.

Identificación de las preguntas

La identificación de las preguntas tiene la misma finalidad pero para ambos casos distintos métodos para obtenerlas. Los tipos de preguntas que se podrán identificar dentro de una sentencia serán los siguientes:

- Usando en analizador morfosintáctico:
  - Dónde
  - Quién/Quiénes
  - Qué

- Usando el analizador basado en dependencias sintácticas.
  - Dónde
  - Cuándo

5.2.3. Construcción de la pregunta.

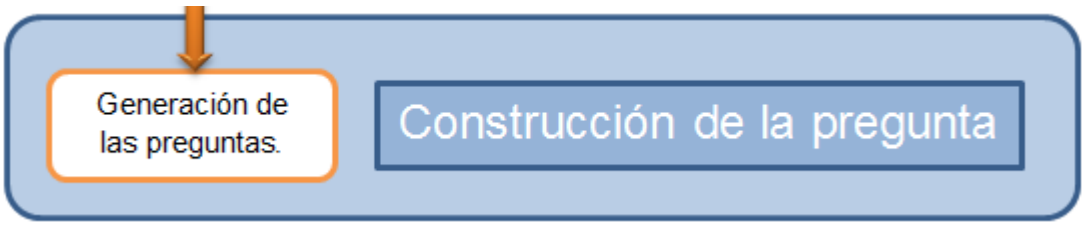


Figura 5-10: Construcción de la pregunta

La última de las etapas es la de construir una pregunta en base a la identificación previa.

Generación de las preguntas

Una vez identificadas las preguntas, se generan las preguntas, para ello nos apoyamos tanto en los resultados guardados de los análisis como en las identificaciones que se han conseguido hacer sobre cada sentencia.

5.3. Generador de preguntas a partir de un analizador morfosintáctico

Para la creación de este generador se usaron las siguientes herramientas:

- Ixa-pipe-tok
- Ixa-pipe-parse

Aparte de estas herramientas de PLN se han creado diversas clases que en combinación con estas han dado la posibilidad de crear el generador. Este generador está basado en el sintagma nominal, esto quiere decir que una vez que la sentencia esté analizada, se buscan todos los sintagmas nominales que se encuentren en ella y dependiendo de distintas condiciones se crearán distintas preguntas. Esta idea fue tomada del artículo sintetizado en el apartado 3.4.

A continuación se explicará detalladamente todos los pasos, basándonos en la arquitectura, que se han dado para su creación.

5.3.1. Selección del objetivo

Selección del texto

El primer paso fue listar por la salida del sistema una lista enumerada con todos los textos disponibles, mediante el cual el usuario podría seleccionar uno de ellos para su posterior tratamiento.

```
Seleccione, por medio de un número, un texto a analizar.
1.- Batalla búlgaro-croata de 927.txt
2.- Batalla de Abrito.txt
3.- Batalla de Adrianópolis.txt
4.- Batalla de Argentovaria.txt
5.- Batalla de Cartagena.txt
6.- batalla de Cuarte.txt
7.- Batalla de Edesa.txt
8.- Batalla de Issos.txt
9.- Batalla de la isla Saltés.txt
10.- Batalla de las Navas de Tolosa.txt
11.- Batalla de Mons Seleucus.txt
12.- Batalla de Mursa Major.txt
13.- Batalla de Pollentia.txt
14.- Batalla de Rímini.txt
15.- Batalla de Solicinium.txt
16.- Batalla de Tapae.txt
17.- Batalla de Uji (1180).txt
18.- Batalla del Puente Milvio.txt
```

Figura 5-11: Salida del sistema, selección del texto.

El programa solo admite números que se encuentren dentro del rango aceptable, este está delimitado por la cantidad de textos que se encuentren en la carpeta donde estos estén.

Selección de una sentencia

Este proceso es automático, se van seleccionando una a una las distintas sentencias del texto, para ello el sistema realiza lo siguiente:

- Tokenizar el texto seleccionado mediante la herramienta ixa-pipe-tokenizer, esto nos permitirá tener las sentencias tokenizadas y separadas.
- Como se puede ver en el siguiente gráfico, el sistema de salida muestra el título del texto elegido, también indica el número de sentencias encontradas en ese texto. Las

sentencias tokenizadas son las que están dentro de los corchetes, cada palabra de estas están separadas por medio de una coma.

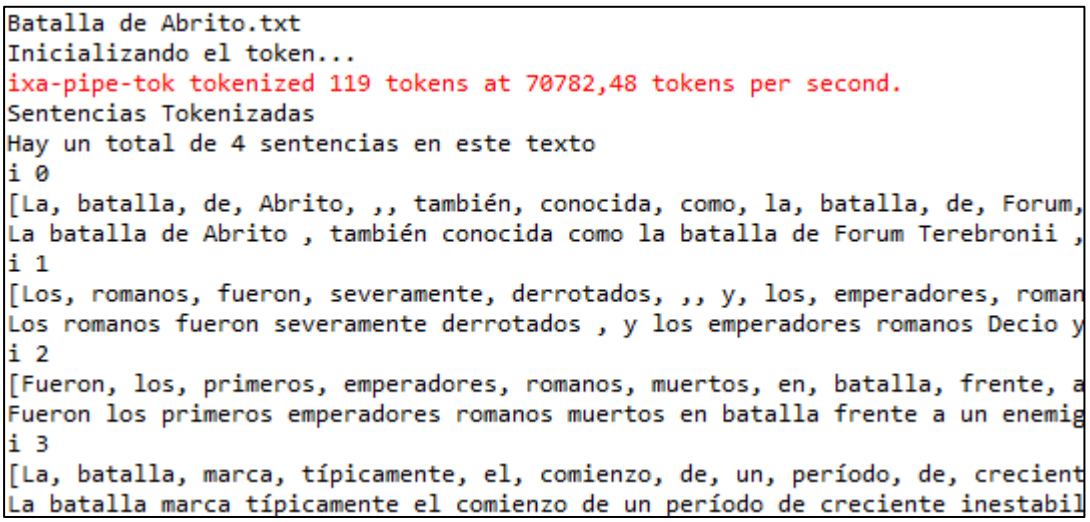


Figura 5-12: Texto tokenizado

Cada una de estas sentencias, como se ha comentado en el apartado de arquitectura, recibirá los análisis posteriores. La idea es que al final se dé como resultado todas las preguntas que se puedan generar a partir de un texto.

Análisis de la sentencia

El analizador que se usa en este generador es el morfosintáctico, este nos viene dado gracias a la herramienta ixa-pipe-parse. Para ello se realiza lo siguiente:

- Una vez que se tenga una de las sentencias seleccionadas se deberá realizar su análisis por medio de la herramienta ixa-pipe-parse, como se indica en el apartado 4.1.4.
- En el siguiente gráfico se observa el tipo de análisis que realiza la herramienta. En caso de que el análisis sea correcto se nos informará de ello, en caso contrario también se nos avisará.

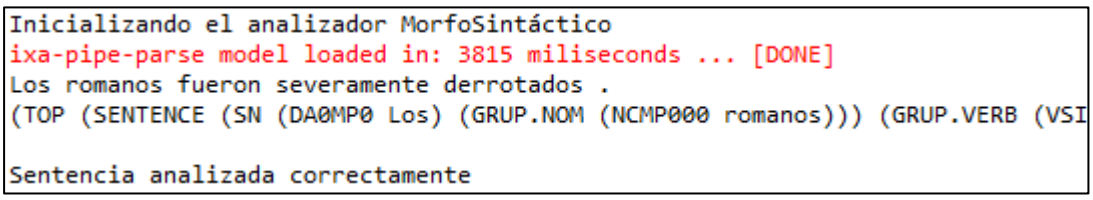


Figura 5-13: Salida del sistema, análisis morfosintáctico.

Guardar resultados

Una vez realizado el análisis este es guardado para sus usos posteriores, para ello se creó una estructura de datos que fuese útil a la hora de acceder a cualquier posición del análisis y que al mismo tiempo fuese ajustable a cualquier tipo de análisis. La estructura es una clase llamada “sentenciaAnalizada”, en los siguientes puntos se profundizará en su funcionamiento.

- Cuenta con tres variables que son las siguientes:

```
private String tipo;
private String texto;
private SentenciaAnalizada nueva[];
```

- Tipo: almacena el valor sintáctico o morfológico de un texto analizado.
- Texto: almacena el texto vinculado con el tipo de un texto analizado.
- Nueva: esta lista del tipo sentenciaAnalizada continuará buscando si dentro de un análisis puede haber otros.

- Mediante el siguiente ejemplo se puede apreciar su funcionamiento:

  Dada la frase *"La batalla aconteció en Roma."*

  El análisis morfosintáctico nos daría el siguiente resultado

```
(TOP (SENTENCE (SN (DA0FS0 La) (GRUP.NOM (NCFS000 batalla)))
(GRUP.VERB (VMIS3S0 aconteció)) (SP (PREP (SPS00 en)) (SN (GRUP.NOM
(NP00000 Roma)))) (FP .)))
```

Mediante la función getChildren de la herramienta ixa-pipe-parse vista en el punto 4.1.4 nos situamos en "SENTENCE"

```
(SENTENCE (SN (DA0FS0 La) (GRUP.NOM (NCFS000 batalla))) (GRUP.VERB
(VMIS3S0 aconteció)) (SP (PREP (SPS00 en)) (SN (GRUP.NOM (NP00000
Roma)))) (FP .))
```

Vista desde otra perspectiva se puede ver que una parte del árbol quedaría así

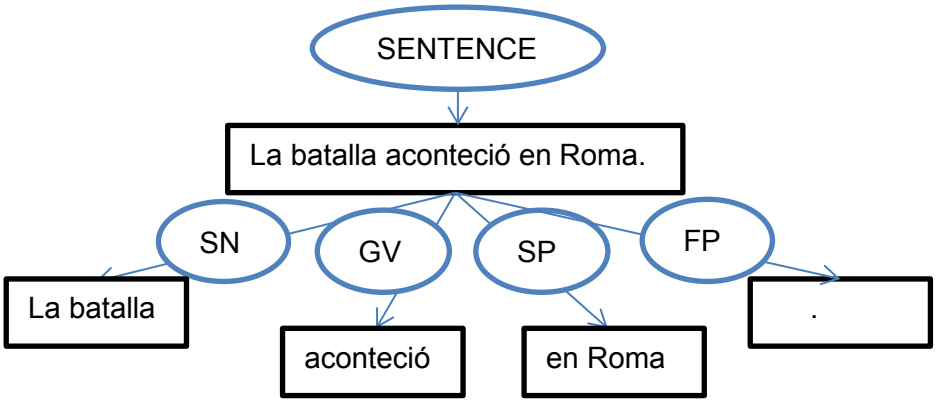


Figura 5-14: árbol morfosintáctico

Y decimos que una parte porque el análisis continua hasta darnos algo parecido a la figura 2-2.

Llegados a este punto sabemos que tenemos que crear una lista de la estructura de datos propuesta, sentenciaAnalizada, con un tamaño de cuatro, ya que la frase está dividida en cuatro partes. En cada uno de los índices estarán guardados los siguientes elementos:

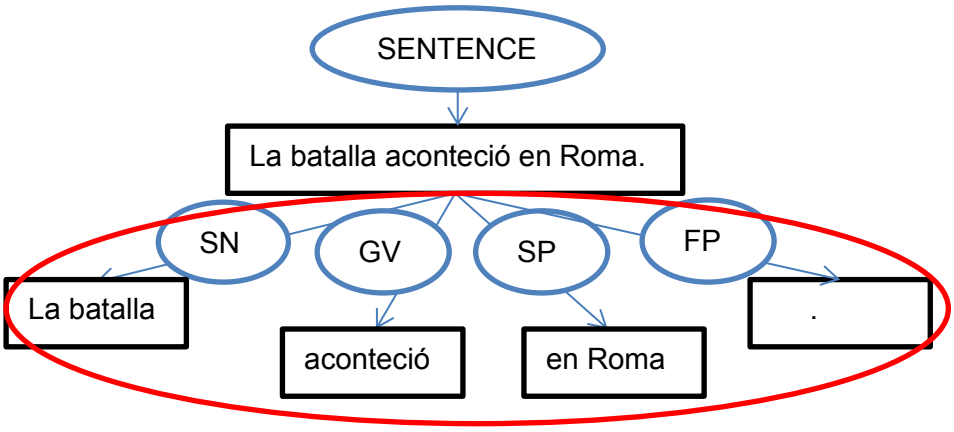


Figura 5-15: Índices seleccionados del análisis morfológico.

- Índice 1 – Tipo “SN” y Texto “La batalla”
- Índice 2 – Tipo “GV” y texto “aconteció”
- Índice 3 – Tipo “SP” y texto “en Roma”
- Índice 4 – Tipo “FP” y texto “.”

Dentro de cada índice, dependiendo si el análisis continua, el array nueva que es una lista de la clase sentenciaAnalizada, seguirá guardando los datos.

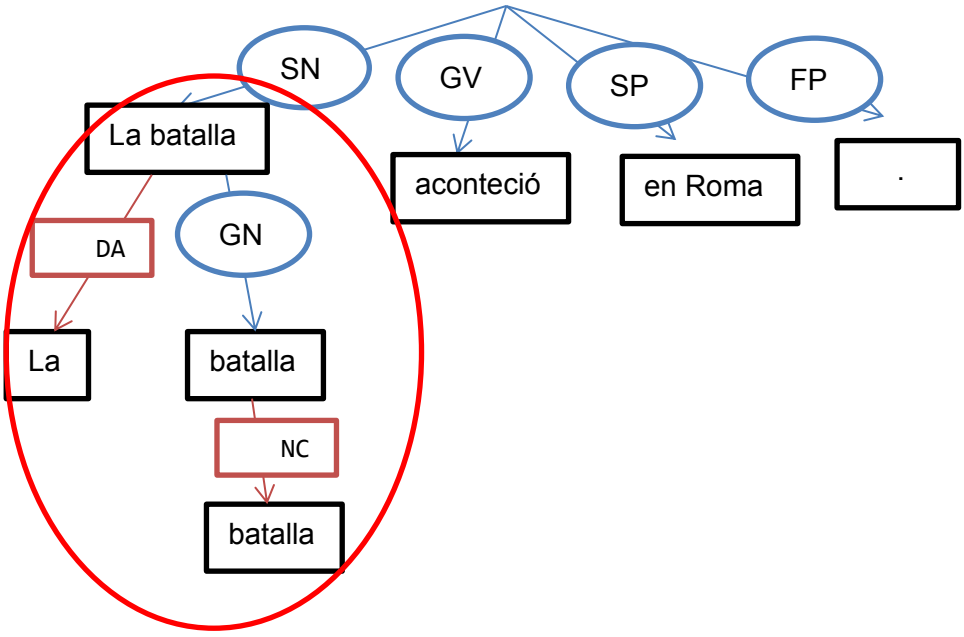


Figura 5-16: índices seleccionados del análisis morfológico II.

- Índice 1 – tipo “SN” y texto “La batalla”
  - Índice 1 – tipo “DA0FS0”, texto “La”
  - Índice 2 – tipo “GN”, texto “batalla”
    - Índice 1 – tipo “NCFS000”, texto “batalla”
- Índice 2 – Tipo “GV” y texto “aconteció”

Básicamente este sería su funcionamiento y su forma de guardar los datos.

- En el momento del guardar los datos en la estructura en la salida del sistema nos saldrá una vista de cómo han quedado guardado los datos.

```
Guardando datos en la estructura...

Tipo: SN  Texto: la batalla
--- Tipo: DA0FS0  Texto: la
--- Tipo: GRUP.NOM  Texto: batalla
------ Tipo: NCFS000  Texto: batalla
Tipo: GRUP.VERB  Texto: acabó
--- Tipo: VMIS3S0  Texto: acabó
Tipo: FP  Texto: .

Datos guardados correctamente
```

Figura 5-17: Salida del sistema de la estructura guardada.

5.3.2. Selección del tipo de pregunta

La selección del tipo de pregunta viene dada dentro de una clase llamada generadorPreguntas, esta clase aparte de seleccionar el tipo de pregunta también se encarga de construir la pregunta. La selección del tipo de preguntas es llamada dentro de una función de esta clase, ya que es necesario saber el tipo de pregunta a la hora de construir la pregunta. Por ese motivo en esta sección se explicará la parte vinculada principalmente a la selección del tipo de pregunta, y en la siguiente se volverá a mencionar el resto de funciones que tiene la clase generadorPreguntas.

- La clase generadorPreguntas cuenta con dos tipos de variables, las principales y las auxiliares, es de las principales de las que hablaremos a continuación.

```
//variables principales
private SentenciaAnalizada sA[];
private BuscadorTiposPalabras buscador;

private Pregunta preguntas[];
private ClasificadorPI clasificador;
```

  - sA: es una variable del tipo SentenciaAnalizada.
  - buscador: se define una variable llamada buscador del tipo perteneciente a la clase BuscadorTiposPalabras. Esta clase tiene la propiedad de buscar dentro de una sentencia analizada un tipo sintáctico o morfológico concreto.
  - Preguntas: En esta variable del tipo Pregunta se pueden almacenar los tipos de las preguntas, las preguntas generadas y las respuestas de estas preguntas.
  - Clasificador: se define esta variable del tipo ClasificadorPI que tiene la capacidad de clasificar las preguntas.

- Esta clase es llamada después de que guardemos las sentencias analizadas en la estructura del tipo SentenciaAnalizada.

```
System.out.println("\nGenerador de preguntas");
genPreg = new GeneradorPreguntas(sA);
genPreg.algPreg();
```

- Después de definir la variable genPreg del tipo GeneradorPreguntas, instanciamos la clase e invocamos al constructor, este recibirá como parámetro la sentencia analizada previamente.

- A continuación llamamos a la función algPreg, esta es la función principal de esta clase. Desde aquí se llaman tanto al clasificador que realizará la selección del tipo de pregunta como al constructor de la pregunta.

  - Previamente a la llamada del clasificador y del constructor se buscan todos los sintagmas nominales que se encuentren en la sentencia analizada.
  - Se depura esta búsqueda mirando si al lado de los sintagmas nominales se encuentra un grupo verbal o un verbo.

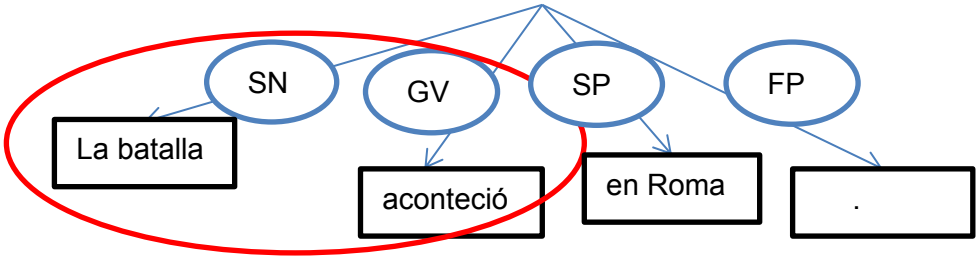


Figura 5-18: Sintagma nominal y grupo verbal.

Estas decisiones son parte del diseño basados en un estudio el cual nos muestra la cantidad de preguntas que se pueden generar mediante una estructura que empieza por un sintagma nominal y a continuación un grupo verbal.

En el siguiente cuadro se puede apreciar una parte resumida del estudio, en donde hay ejemplos de sentencias completas, seguidas de un análisis, donde se muestran que a partir de una estructura que empieza por un sintagma nominal seguido de un grupo verbal se puede formar una pregunta.

Para este estudio se tomaron diversas sentencias al azar de diversos textos, este estudio se llevó a cabo antes de generar cualquier algoritmo.

Sentencias	Análisis rápido	Pregunta	Respuesta
Los almorávides iniciaron las hostilidades el 14 de octubre.	SN Los almorávides GRUP.VERB iniciaron SN las hostilidades SN el 14 de octubre FP .	¿Cuándo iniciaron las hostilidades los almorávides?	El 14 de octubre.
La batalla acabó sin muchas bajas.	SN la batalla GRUP.VERB acabó SP sin muchas bajas FP .	¿Qué acabó sin muchas bajas?	La batalla.
El gobernante búlgaro, Simeón, era un hombre sabio y capaz, con un espíritu inquieto e insaciable.	SN El gobernante búlgaro, Simeón , GRUP.VERB era SN un hombre sabio y capaz , con un espíritu inquieto e insaciable FP .	¿Quién era un hombre sabio y capaz, con un espíritu inquieto e insaciable?	El gobernante búlgaro, Simeón.
Constantino y su ejército estaban persiguiendo algunos sármatas que habían cruzado el río Danubio al territorio de Licinio.	SN Constantino y su ejército GRUP.VERB estaban persiguiendo SN algunos sármatas que habían cruzado el río Danubio al territorio de Licinio FP .	¿Quiénes estaban persiguiendo algunos sármatas que habían cruzado el río Danubio al territorio de Licinio?	Constantino y su ejército.
El 3 de febrero fue la fecha que acabó la guerra.	SN El 3 de febrero GRUP.VERB fue SN la fecha que acabó la guerra	¿Cuándo fue la fecha que acabó la guerra?	El 3 de febrero.

Tabla 5-2: Frases analizadas que contienen un SN y GV

- Si se han encontrado SN seguidos de GV estos se guardan en una lista. Es a partir de aquí donde se empieza la selección de la pregunta, seguido de la construcción de esta.

Identificación de las preguntas

Una vez tengamos las sentencias analizadas, la lista con la información de donde se encuentran dentro de estas sentencias las estructuras formadas por un SN y un GV, llega el momento de realizar la clasificación de las preguntas.

Este clasificador tendrá la capacidad de clasificar tres tipos de preguntas, **Quién/Quiénes, Qué y Cuándo**, el algoritmo de clasificación para cada una de las preguntas es distinto.

En este apartado se explicarán los distintos algoritmos usados para la clasificación de las preguntas.

- **Algoritmo para identificar la pregunta del tipo Qué.**

  - Buscar si en la sentencia analizada hay un determinante artículo femenino en el SN principal, el que está a la izquierda del GV.
    - Si se cumple esta condición entonces es una pregunta del tipo Qué.
  - Si no se encuentra ningún determinante artículo femenino entonces se deberá buscar dentro del SN el Grupo Nominal.
  - Buscar dentro del grupo nominal el nombre (núcleo).
    - Si no se encuentra un nombre dentro del grupo nominal la sentencia está mal formada.
    - Si el nombre tiene la propiedad de ser un sustantivo común, entonces es una pregunta del tipo Qué.
  - Si no cumple ninguna de las propiedades anteriores es que no es una pregunta del tipo Qué.

Ejemplo:

1. *La batalla de Abrito terminó.*

```
Tipo: SN  Texto: La batalla de Abrito
--- Tipo: DA0FS0  Texto: La
--- Tipo: GRUP.NOM  Texto: batalla de Abrito
------ Tipo: NCFS000  Texto: batalla
------ Tipo: SP  Texto: de Abrito
--------- Tipo: PREP  Texto: de
---------- Tipo: SPS00  Texto: de
--------- Tipo: SN  Texto: Abrito
---------- Tipo: GRUP.NOM  Texto: Abrito
--------------- Tipo: NP00000  Texto: Abrito
Tipo: GRUP.VERB  Texto: terminó
--- Tipo: VMIS3S0  Texto: terminó
Tipo: FP  Texto: .
```

Nombre común de un Grupo Nominal dentro de un Sintagma nominal. Condición necesaria para formar una pregunta del tipo Qué.

Figura 5-19: Ejemplo de análisis morfosintáctico por la salida del sistema (1).

En el Apéndice 2 se puede ver el código generado para describir el algoritmo para identificar la pregunta del tipo Qué.

La evaluación para apoyar este algoritmo viene dada en el apartado 6.1.1.

- **Algoritmo para identificar la pregunta del tipo Quién/Quiénes.**

- Buscar si en la sentencia analizada en el Sintagma nominal que está a la izquierda del Grupo Verbal hay una conjunción.
  - Si se da el caso es una pregunta del tipo Quiénes.
- Buscar en la sentencia analizada si hay un determinante artículo masculino o un determinante posesivo masculino.
  - Si se da el caso es una pregunta del tipo Quién.
- Si no se encuentra ninguno de los anteriores entonces se deberá buscar dentro del SN el Grupo Nominal.
- Buscar dentro del grupo nominal el nombre (núcleo).
  - Si no se encuentra un nombre dentro del grupo nominal la sentencia está mal formada.
  - Si e nombre está un sustantivo propio entonces es una pregunta del tipo Quién.
- Si no cumple ninguna de las propiedades anteriores es que no es una pregunta del tipo Quién/Quiénes.

Ejemplo:

1. *El comandante de las fuerzas búlgaras en esta batalla fue el Duque Alogobotur.*

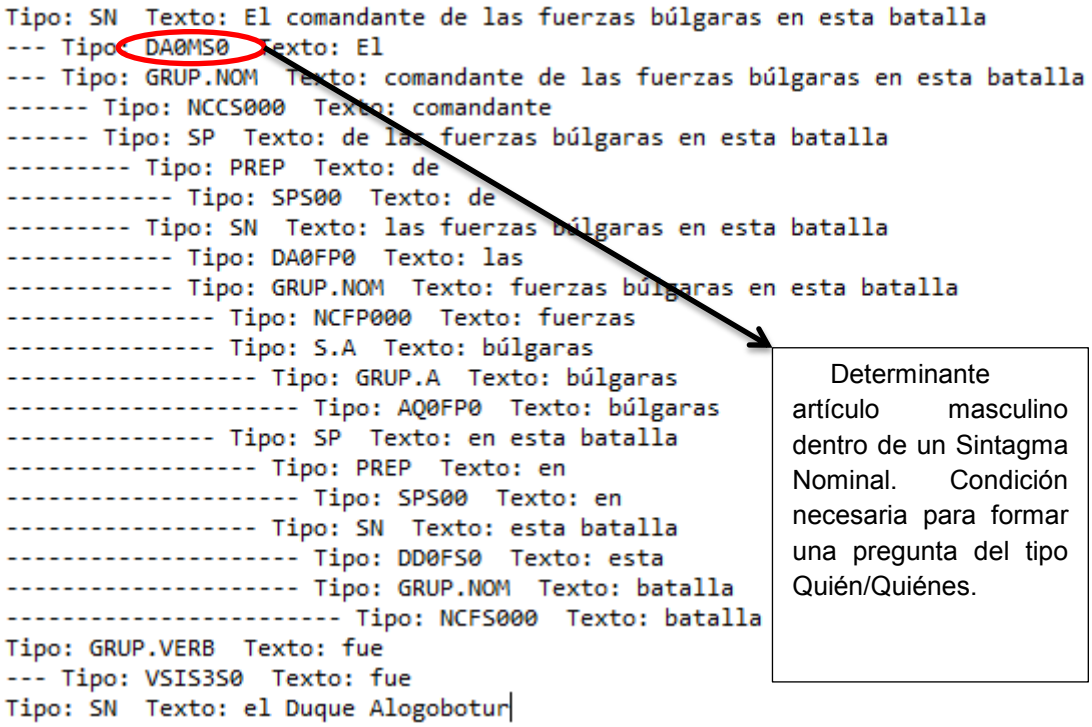


Figura 5-20: Ejemplo de análisis morfosintáctico por la salida del sistema (2).

En el Apéndice 3 se puede ver el código generado para describir el algoritmo para identificar la pregunta del tipo Quién/Quiénes.

La evaluación para apoyar este algoritmo viene dada el en el capítulo 6.1.1.

- **Algoritmo para identificar la pregunta del tipo Cuándo.**

  - Buscar en la sentencia analizada si hay una fecha (tipo w) de cualquier tipo.
    - Si hay una fecha en algún lugar de la sentencia analizada entonces es una pregunta de tipo Cuándo.

- Si no se encuentra ninguna fecha entonces no es una pregunta del tipo Cuándo.

Ejemplo:

1. *Los almorávides iniciaron las hostilidades el 14 de octubre.*

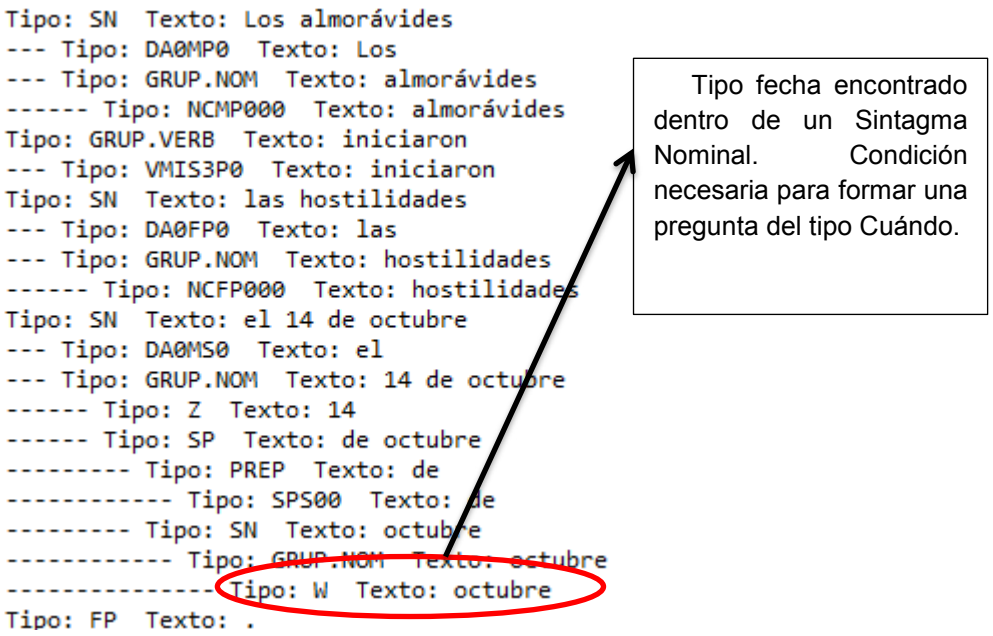


Figura 5-21: Ejemplo de análisis morfosintáctico por la salida del sistema (3).

En el Apéndice 4 se puede ver el código generado para describir el algoritmo para identificar la pregunta del tipo Cuándo.

La evaluación para apoyar este algoritmo viene dada el en el capítulo 6.1.1.

Todos estos resultados se almacenan en la variable pregunta que es del tipo Pregunta. Dentro de esta clase como ya se ha explicado un poco más arriba se pueden almacenar los tipos de las preguntas.

5.3.3. Construcción de la pregunta

En el anterior punto de selección del tipo de pregunta se comentó que la clase generadorPreguntas procedía en un primer momento a realizar la identificación de las preguntas, después de tener todas las preguntas identificadas es cuando se dispone a armar las preguntas.

```
private void armarPreguntas() {
    //qué
    if (clasificador.getPInt()[0] == 1){
        generarPreguntaQue();
    }
    //quién-quiénes
    if (clasificador.getPInt()[1] == 1){
        generarPreguntaQuien();
    }
    //cuándo
    if (clasificador.getPInt()[2] == 1){
        generarPreguntaCuando();
    }
}
```

Generación de las preguntas

Cada pregunta se construye de una manera distinta, a continuación se específica la construcción para cada una de las preguntas.

- Generación de la pregunta Qué.

  - Se empezará la pregunta con el símbolo de interrogación "¿" seguido de la palabra "Qué".
  - Se deberá mirar si al lado izquierdo del Sintagma Nominal (SN), que tiene a su lado derecho un Grupo Verbal, hay otro Sintagma Nominal (SNI).
    - Si así es, se deberá comprobar que el SN es un pronombre, en cuyo caso formará parte de la pregunta, concretamente irá después de la palabra "Qué".
  - Se añadirá todo el texto que este entre el Grupo Verbal y el final de la sentencia. Se terminará con el símbolo "?".

Ejemplo:

1. *La batalla de Mons Seleucus ocurrió en el año 353 entre las fuerzas de Constancio II y las fuerzas del usurpador Magnencio.*

```
SN La batalla de Mons
SN Seleucus
GRUP.VERB ocurrió
SP en el año 353
SP entre las fuerzas de Cons
SN II y las fuerzas del usurpador Magnencio
FP .
```

Sentencia con la estructura SN SN GV, donde dentro del primer SN encontramos a un determinante artículo femenino y donde el SN del medio no es un pronombre.

Figura 5-22: Análisis morfosintáctico rápido para determinar el tipo de pregunta Qué.

Pregunta generada: ***¿Qué ocurrió en el año 353 entre las fuerzas de Constancio II y las fuerzas del usurpador Magnencio?***

- Generación de la pregunta Quién/Quiénes.

  - Se empezará la pregunta con el símbolo de interrogación "¿" seguido de la palabra "Quién/Quiénes" dependiendo de cuál de las dos sea.
  - Se deberá mirar si al lado izquierdo del Sintagma Nominal (SN), que tiene a su lado derecho un Grupo Verbal, hay otro Sintagma Nominal (SNI).

- Si así es, se deberá comprobar que el SN es un pronombre, en cuyo caso formará parte de la pregunta, concretamente irá después de la palabra "Quién/Quiénes".

➢ Se añadirá todo el texto que este entre el Grupo Verbal y el final de la sentencia. Se terminará con el símbolo "?".

Ejemplo:

1. *Su familia le envió a Aarau para terminar sus estudios secundarios en la escuela cantonal de Argovia.*

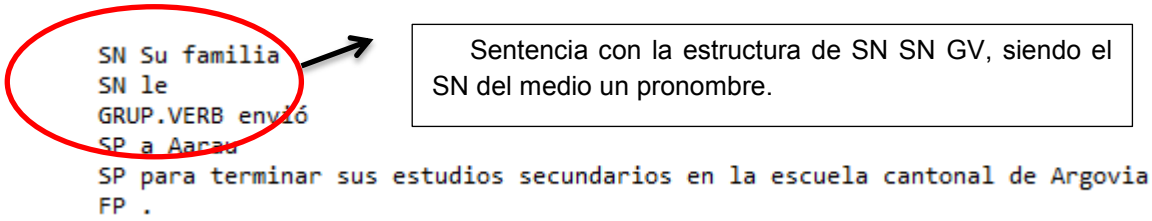


Figura 5-23: Análisis morfosintáctico rápido para determinar el tipo de pregunta Quién.

Pregunta generada: ***¿Quién le envió a Aarau para terminar sus estudios secundarios en la escuela cantonal de Argovia?***

- Generación de la pregunta Cuándo.

➢ Se empezará la pregunta con el símbolo de interrogación "¿" seguido de la palabra "Cuándo" .

➢ Dependiendo de si el SN que está seguido de un Grupo Verbal es un pronombre se deberá:

- Si es un pronombre se incorporará el texto por el cual esté formado a la pregunta, se seguirán incluyendo las palabras hasta detectar que se ha encontrado el tipo en el cuál se incluye la fecha.
- Si no es un pronombre, nos situaremos en el Grupo Verbal e incluiremos el texto por el cual está formado a la pregunta, se seguirán incluyendo las palabras hasta detectar que se ha encontrado el tipo en el cuál se incluye la fecha, seguidos de esto se incorporará el Sintagma Nominal que estaba seguido de un Grupo Verbal.

➢ Al final de la sentencia. Se terminará con el símbolo "?".

Ejemplo:

1. *La batalla de Abrito, también conocida como la batalla de Forum Terebronii , aconteció en la provincia romana de Mesia Inferior probablemente el 01 de julio de 251 , entre el Imperio romano y una federación de miembros de una tribu escita bajo el rey godo Cniva .*

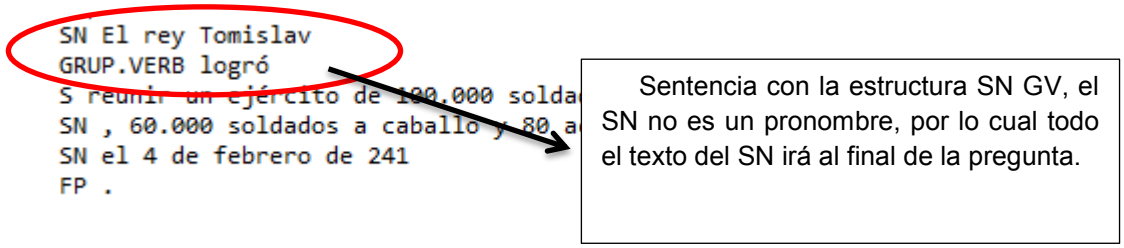


Figura 5-24: Análisis morfosintáctico rápido para determinar el tipo de pregunta Dónde.

Pregunta generada: ***¿Cuándo logró reunir un ejército de 100.000 soldados a pie, 60.000 soldados a caballo y 80 acorazados, el rey Tomislav?***

5.4. Generador de preguntas a partir de un analizador basado en dependencias sintácticas.

Este generador a diferencia del otro analizador que puede tener como entrada un texto formado por varias sentencias, tiene como entrada solo una sentencia. Y es a partir de esta sentencia que se intentará generar la pregunta.

Para la creación de este generador se usaron las siguientes herramientas:

- lxa-pipe-tok
- lxa-pipe-pos
- lxa-pipe-srl

A parte de estas herramientas PNL se han creado diversas clases que en combinación con estas han dado la posibilidad de crear el generador.

El análisis basado en dependencias sintácticas realiza un análisis sintáctico que nos permite situar diversos elementos que el otro analizador no nos permitía como es el caso del sujeto o del complemento circunstancial.

Hay que señalar que si bien es cierto que el análisis sintáctico es más exhaustivo que en caso del analizador morfosintáctico, todavía quedan algunos campos por ser terminados, por ejemplo en el caso del complemento circunstancial, es cierto que el analizador los detecta, pero no especifica si son del tipo temporal, de lugar etc…

A continuación se explicará detalladamente todos los pasos, basándonos en la arquitectura, que se han dado para su creación.

5.4.1. Selección del objetivo

Selección del texto

Este paso es idéntico al realizado en el generador de preguntas basado en un análisis morfosintáctico, se listan todos los posibles textos por la salida del sistema y el usuario selecciona uno de ellos

```
198.- La batalla del Gran Zab.txt
199.- La batalla del Indo.txt
200.- La batalla del Lago Peipus.txt
201.- La Batalla del Puente de Ludford.txt
202.- La batalla del río Vorskla.txt
203.- La batalla del Salado.txt
204.- La Batalla del Sava.txt
205.- La batalla del Vado de Jacob.txt
206.- La Batalla junto al río Eno.txt
207.- La batalla junto al río Temes.txt
208.- La batalla naval de Formigues.txt
209.- La batalla naval de Yamen.txt
210.- La primera batalla de Homs.txt
211.- La Primera Batalla de St Albans.txt
212.- La revuelta de Tiro.txt
213.- la segunda batalla de Albelda.txt
214.- La Segunda Batalla de St Albans.txt
2
```

Figura 5-25: Salida del sistema, selección del texto (II)

Selección de una sentencia

Una vez obtenido el texto debemos seleccionar una sentencia, para ello el sistema nos muestra por salida todas las sentencias del texto y nos indica que debemos seleccionar una de ellas.

Para poder tener las sentencias separadas se debe tokenizar las palabras que forman el texto. Una vez que se tenga tokenizado se deberá almacenar el resultado tokenizado en una variable que represente la estructura del formato NAF la cual nos viene dada gracias a la librería kaflib. Esta variable nos permitirá listar por el sistema las sentencias:

```
Seleccione, por medio de un número, la Sentencia que desea analizar.

1 [La, batalla, de, Abrito, ,, también, conocida, como, la, batalla, de, F

2 [Los, romanos, fueron, severamente, derrotados, ,, y, los, emperadores,

3 [Fueron, los, primeros, emperadores, romanos, muertos, en, batalla, fren

4 [La, batalla, marca, típicamente, el, comienzo, de, un, período, de, cre
```

Figura 5-26: Salida del sistema, selección de una sentencia

Al seleccionar una sentencia, se deberá crear un fichero temporal en formato txt en el cual se almacene su contenido.

Este contenido se volverá a tokenizar y una vez tokenizado se volverá a almacenar (annotate) el resultado en otra variable del tipo NAF.

```
//kaf (NAF)
kafNuevo = new CargarKaf("es","1.1.12");

//token
token = new CargarToken(dirSentencia );
token.getAnnotate().tokenizeToKAF(kafNuevo.getKaf());
```

Análisis de la sentencia

Una vez que tengamos la sentencia seleccionada tokenizada y almacenada en la variable NAF, se deberá usar la herramienta ixa-pipe-pos para realizar un etiquetado gramatical. Al igual que antes su resultado se deberá almacenar en la variable NAF.

```
//pos
CargarPOS pos = new CargarPOS();
pos.getAnnotate().annotatePOSToKAF(kafNuevo.getKaf());
```

A continuación se deberá usar la herramientra ixa-pipe-srl, especificando que se desea usar la opción de dependencias.

```
//srl
CargarSRL srl = new CargarSRL("spa","only-deps");
srl.getAnnotate().SRLToKAF(kafNuevo.getKaf(), "spa","only-deps");
```

Una de las funciones de la herramienta ixa-pipe-srl, permite obtener la lista de dependencias, es dentro de esta lista donde tendremos todo el análisis que la propia herramienta realiza de manera automática.

```
//obtener lista de Dependencias
List<Dep> listaDep =  kafNuevo.getKaf().getDeps();
```

En el siguiente ejemplo se muestra por la salida del sistema la lista de dependencias.

1. *Los romanos fueron severamente derrotados, y los emperadores romanos Decio y su hijo Herenio Etrusco resultaron ambos muertos en la batalla.*

```
ITERACION : 0 -----spec(romanos, Los)
ITERACION : 1 -----suj(fueron, romanos)
ITERACION : 2 -----cc(derrotados, severamente)
ITERACION : 3 -----atr(fueron, derrotados)
ITERACION : 4 -----f(fueron, ,)
ITERACION : 5 -----coord(fueron, y)
ITERACION : 6 -----spec(emperadores, los)
ITERACION : 7 -----suj(resultaron, emperadores)
ITERACION : 8 -----s.a(emperadores, romanos)
ITERACION : 9 -----sn(emperadores, Decio)
ITERACION : 10 -----coord(emperadores, y)
ITERACION : 11 -----spec(hijo, su)
ITERACION : 12 -----sn(emperadores, hijo)
ITERACION : 13 -----sn(hijo, Herenio)
ITERACION : 14 -----sn(Herenio, Etrusco)
ITERACION : 15 -----S(fueron, resultaron)
ITERACION : 16 -----spec(muertos, ambos)
ITERACION : 17 -----cd(resultaron, muertos)
ITERACION : 18 -----cc(resultaron, en)
ITERACION : 19 -----spec(batalla, la)
ITERACION : 20 -----sn(en, batalla)
ITERACION : 21 -----f(fueron, .)
```

Figura 5-27: Salida del sistema, análisis basado en dependencias sintácticas.

En las figuras 2-3 y 2-4 del apartado 2.3 se muestra un ejemplo más esclarecedor, que puede servir para un mayor entendimiento.

Guardar resultados

Una gran ventaja de usar una variable que representa la estructura del formato NAF es que no se necesita usar una estructura extra para almacenar los datos y tratarlos. La propia estructura cuenta con diversas funciones que hacen que el tratamiento de los datos sea sencillo.

5.4.2. Selección del tipo de pregunta.

La selección del tipo de pregunta viene dada dentro de una clase llamada generadorPreguntas que cumple el mismo cometido que el anterior generador, recordar que el generador mediante un análisis morfosintáctico y el de basado en dependencias son distintos programas, muchas de sus clases tienen el mismo nombre ya que su arquitectura es muy parecida.

Al igual que en el anterior generador la selección del tipo de preguntas es llamada dentro de una función de esta clase llamada algPreg.

- A continuación se explicarán las variables más importantes y destacables de esta clase.

```
private List<Dep> listaDep;
private Pregunta preguntas[];
ClasificadorPI clasificador;
```

- ListaDep: En esta variable se encuentran la lista de las dependencias.
- Preguntas: En esta variable del tipo Pregunta se pueden almacenar los tipos de las preguntas, las preguntas generadas y las respuestas de estas preguntas.
- Clasificador: se define esta variable del tipo ClasificadorPI que tiene la capacidad de clasificar las preguntas.

- Esta clase es llamada después de que se realice el análisis de la sentencia y se le pasa como parámetro la lista de dependencias.

```
//Generador de Preguntas
GeneradorPreguntas genPreg = new GeneradorPreguntas(listaDep);
genPreg.algPreg();
```

Como ya se ha dicho dentro de esta clase se llama a la función que clasificará a la sentencia dentro de un tipo de pregunta. Hay que destacar que ambas clasificaciones están basadas en la detección del complemento circunstancial. El único problema es que el complemento circunstancial que detecta la herramienta es genérico, no especifica para cada caso el tipo de complemento circunstancial. Por ello se ha tenido que complementar este análisis para poder determinar qué tipo de pregunta se puede obtener.

Identificación de la pregunta

Una vez hallamos obtenido la lista de dependencias sintácticas de la sentencia es el momento de identificar la pregunta.

Este clasificador tendrá la capacidad de clasificar dos tipos de preguntas, **Dónde y Cuándo**, el algoritmo de clasificación para cada una de las preguntas es distinto.

En este apartado se explicarán los distintos algoritmos usados para la clasificación de las preguntas.

- **Algoritmo para identificar la pregunta del tipo Dónde.**

  - Se deberá buscar si dentro de la lista de dependencias hay alguna función sintáctica del tipo complemento circunstancial.
    - Si se cumple esta condición y encuentra algún complemento circunstancial se deberá comprar si ese complemento circunstancial es de lugar, para ello miramos que su primer elemento por el que está formado es una preposición.
      - Si está formado por una preposición se deberá mirar si el primer elemento de la siguiente función sintáctica vinculada con la preposición no está catalogado como una fecha, en el algoritmo para identificar la pregunta del tipo Cuándo se explica cómo se cataloga un elemento como fecha.
        - Si no está catalogado como una fecha entonces es una pregunta del tipo Dónde.
        - Si es catalogado como una fecha entonces no es una pregunta del tipo Dónde.
      - Si no está formado por ninguna preposición entonces no es una pregunta del tipo Dónde
    - Si no encuentra ningún complemento circunstancial es que no es una pregunta del tipo Dónde.

Ejemplo:

1. *Los romanos fueron severamente derrotados, y los emperadores romanos Decio y su hijo Herenio Etrusco resultaron ambos muertos en la batalla.*

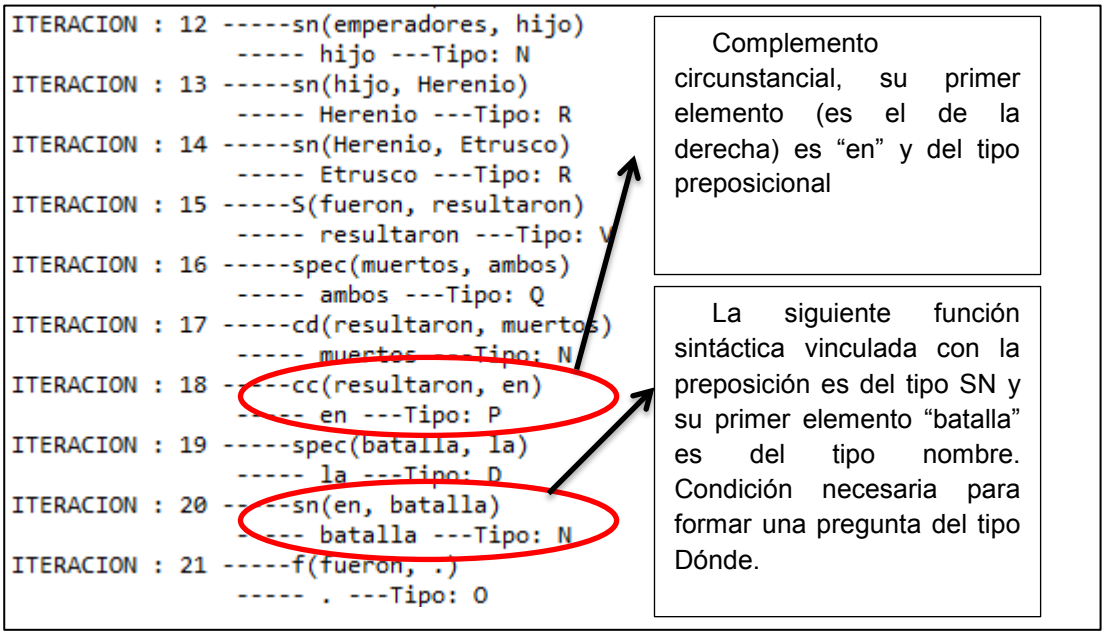


Figura 5-28: Salida del sistema, identificación de la pregunta del tipo Dónde.

A continuación se muestra el trozo relacionado con la detección en forma de árbol para una mayor comprensión.

En el Apéndice 5 se puede ver el código generado para describir el algoritmo para identificar la pregunta del tipo Dónde.

- **Algoritmo para identificar la pregunta del tipo Cuándo.**

  - Se deberá buscar si dentro de la lista de dependencias hay alguna función sintáctica del tipo complemento circunstancial.
    - Si se cumple esta condición y encuentra algún complemento circunstancial se deberá comprar si ese complemento circunstancial es de tiempo, para ello se deberá comprobar si el primer elemento de la siguiente función sintáctica vinculada con el primer elemento del complemento circunstancial está catalogada como una fecha.
      - Un elemento es catalogado como fecha si está formada por:
        - Las siguientes palabras: mes, día, año y fecha
        - Si el elemento es del tipo W (tipo fecha).
      - Si es catalogado como una fecha será entonces un complemento circunstancial de tiempo y por ende será una pregunta del tipo Cuándo.
      - Si no es catalogado como una fecha no será una pregunta del tipo Cuándo.

- Si no encuentra ningún complemento circunstancial entonces no será una pregunta del tipo Cuándo.

Ejemplo:

1. *La batalla de Anquialo tuvo lugar en el año 763, cerca de la ciudad de Pomorie en la costa búlgara del mar Negro.*

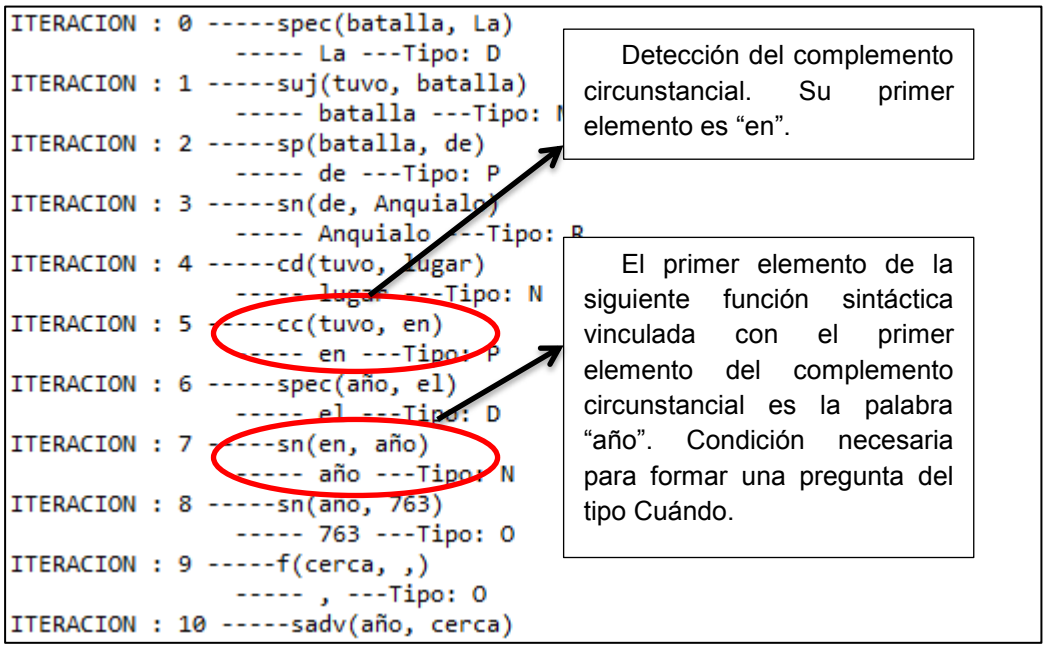
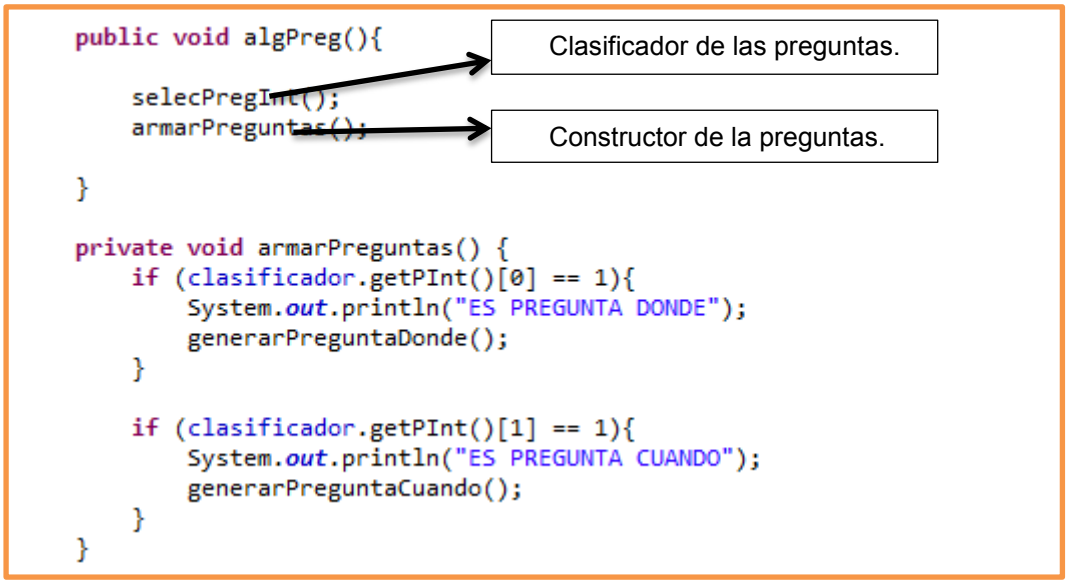


Figura 5-29: Salida del sistema, identificación de la pregunta del tipo Cuándo.

En el Apéndice 6 se puede ver el código generado para describir el algoritmo para identificar la pregunta del tipo Cuándo.

5.4.3. Construcción de la pregunta

Una vez se tengan clasificadas las preguntas que se pueden realizar a partir de una sentencia analizada, se procede a armar las preguntas, como ya se comentó se realiza desde una función de la clase GeneradorPreg, concretamente la función se llama algPreg.




```
public void algPreg(){

    selecPregInt();
    armarPreguntas();

}

private void armarPreguntas() {
    if (clasificador.getPInt()[0] == 1){
        System.out.println("ES PREGUNTA DONDE");
        generarPreguntaDonde();
    }

    if (clasificador.getPInt()[1] == 1){
        System.out.println("ES PREGUNTA CUANDO");
        generarPreguntaCuando();
    }
}
```

Generación de las preguntas

Ambos constructores de pregunta tiene diversos puntos muy similares, dentro de esta similitud cabe destacar dos procedimientos que son prácticamente igual para ambas y que son las que se encargan de generar una frase a partir de diversos argumentos.

Estos procedimientos son:

- situarSujeto

```
private void situarSujeto(int val) {
```

Este procedimiento recibe un parámetro el cual le indica si se está realizando la generación para una pregunta del tipo Dónde o Cuándo.

Su función consiste en buscar dentro de la lista de dependencias al sujeto vinculado con el complemento circunstancial del cual se ha obtenido la selección del tipo de pregunta. Hay una variable global del tipo string llamada palabraP que toma el valor a partir del cual se relaciona al complemento circunstancial durante la ejecución de este procedimiento.

El siguiente ejemplo será representado en forma de lista y en forma de árbol para hacer más sencilla su explicación, como ya se ha explicado en el apartado 2.3 en las figuras 2-3 y 2-4 se puede ver la relación que tiene con una lista de dependencias.

Ejemplo:

1. *La batalla aconteció en Roma.*

Lista de dependencias.

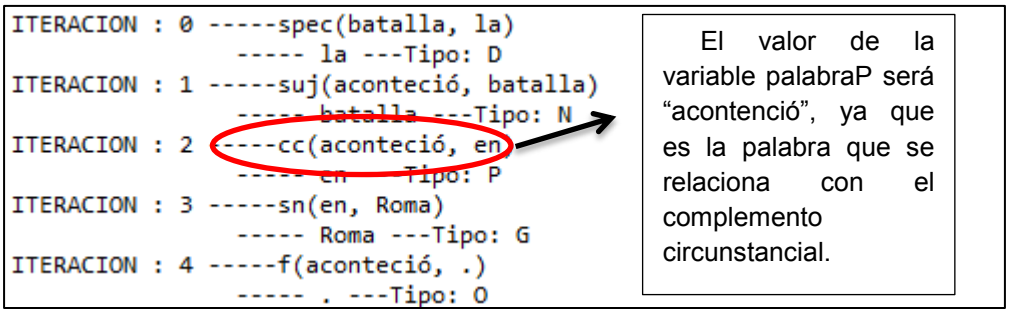


Figura 5-30: Salida del sistema, lista de dependencias.

Lista de dependencias en forma de árbol.

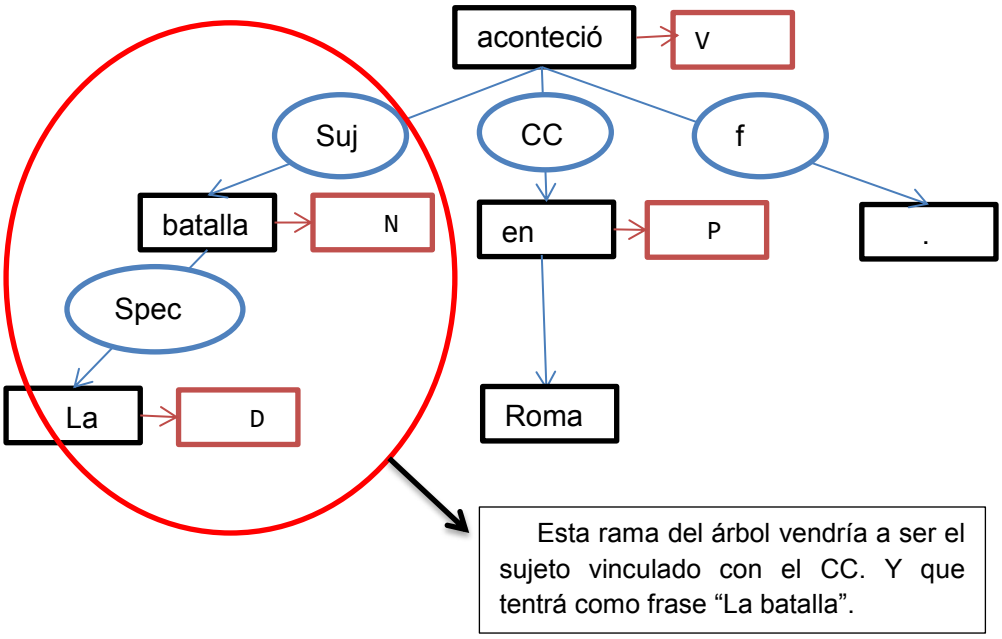


Figura 5-31: Sujeto en un análisis basado en dependencias en forma de árbol.

- formarFrase

```
private String formarFrase(String palPadre, int max,
                           int min, String id, int maxTot) {
```

La función formarFrase es una función recursiva que tiene como objetivo formar frases a partir de los parámetros dados, es llamada después del procedimiento situarSujeto, que es el que se encarga de guardar los parámetros en variables globales que luego serán usados en la llamada de esta función. Estas variables tienen las siguientes propiedades:

- palPadre: indica la palabra en un nodo padre, por ejemplo en la figura 5-31, después de situar el sujeto, la palabra del nodo padre será "batalla".
- max, min: indican donde se tiene que buscar dentro de la lista de dependencias, en la figura 5-30 se puede ver que el sujeto se ha encontrado en la iteración 1, que será el máximo, el mínimo será el 0.

- maxTot: al ser un programa recursivo en cada recursividad se van cambiando todos los parámetros, todos a excepción de maxTot, este indicará siempre cual es el valor max del padre, según el ejemplo de la figura 5-30 será el 1.

Si miramos el árbol, la idea de la recursividad es que la frase se vaya formando a partir de mirar si hay por debajo palabras, y a su vez mirando si por debajo de estas todavía hay palabras que puedan formar la frase.

Para el siguiente ejemplo hemos cogido la parte relacionada al sujeto de la frase.

Ejemplo:

1. *La gran batalla de Abrito aconteció en Roma.*

Lista de dependencias.

```
ITERACION : 0 -----spec(batalla, La)
                  ----- La ---Tipo: D
ITERACION : 1 -----s.a(batalla, gran)
                  ----- gran ---Tipo: G
ITERACION : 2 -----suj(aconteció, batalla)
                  ----- batalla ---Tipo: N
ITERACION : 3 -----sp(batalla, de)
                  ----- de ---Tipo: P
ITERACION : 4 -----sn(de, Abrito)
                  ----- Abrito ---Tipo: R
```

Figura 5-32: Salida del sistema, sujeto dentro de la lista de dependencias.

Lista de dependencias en forma de árbol.

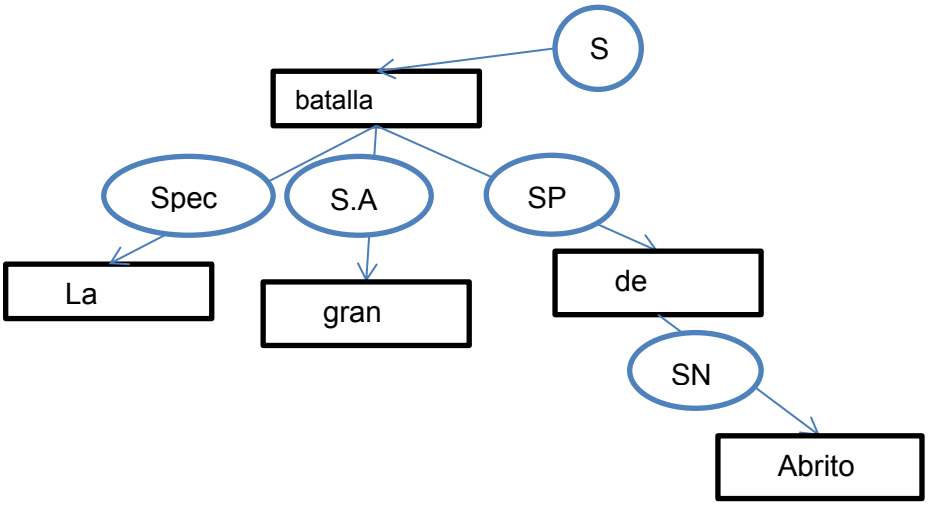


Figura 5-33: Sujeto en una lista de dependencias en forma de árbol.

- Para formar la frase en un primer momento nos situamos el nodo que tiene por nombre “batalla”, esta será la palabra padre.
- La palabra padre tiene 3 nodos hijos. Los índices max y min son los que marcan la búsqueda dentro de la lista de dependencias, en la figura 5-32 podemos ver que la palabra padre “batalla” aparece en la iteración 2, su valor max y min serán 2 y 0 respectivamente. Estos índices nos ayudarán a saber si las palabras que vienen a continuación van a la izquierda o derecha de la palabra padre.
- Seleccionamos el primer nodo hijo empezando por la izquierda, si este no tiene hijos entonces su valor se guardará, en este caso se guardará el valor “La” y sabemos que esté irá a la izquierda de su padre porque si miramos en la figura 5-

32 en la lista de dependencias aparece en la iteración 0, mientras que la palabra padre aparece en la 2, el valor max, min de este nodo hijo será de 0 en ambos casos.

- Seleccionamos el segundo nodo hijo y guardamos su valor "gran" seguido del anterior nodo hijo. Al igual que antes también sabemos que irá a la izquierda del padre ya que su valor max-min es 1, menor que el del padre que es 2.
- En este punto se detecta que el tercer nodo hijo que viene a continuación tiene un valor min-max de 4 y por lo tanto debe ir a la derecha del padre. Antes de realizar esto se agrupa la frase quedando de la siguiente forma "La gran batalla".
- Como se ha detectado que el tercer nodo hijo que tiene la palabra "de" va a la derecha, todo lo que se halle por debajo irá a la derecha de la frase.
- Este tercer nodo hijo tiene a su vez otro hijo que tiene la palabra "Abrito", este es detectado que va a la derecha y como este ya no tiene más nodos por debajo se acaba uniendo a la derecha de su padre, formando la frase "de Abrito".
- Al final se unen las 2 frases generando la frase "La gran batalla de Abrito".

Una vez explicada estas dos funciones que aparecen en ambas generaciones solo queda agregar algunos datos para generar la pregunta.

- Generación de la pregunta Dónde.

  - Se empezará la pregunta con el símbolo de interrogación "¿" seguido de la palabra "Dónde".
  - Se deberá situar el sujeto mediante el procedimiento situarSujeto, en este paso se rellenará la variable global palabraP.
  - Se deberá formar la frase mediante la función formarFrase, se devolverá la frase la cual llamaremos fraseP.
  - Se añadirá a la pregunta palabraP seguida de fraseP.
  - Se terminará añadiendo el símbolo de interrogación "?".

Ejemplo:

1- *Los romanos fueron severamente derrotados, y los emperadores romanos Decio y su hijo Herenio Etrusco resultaron ambos muertos en la batalla .*

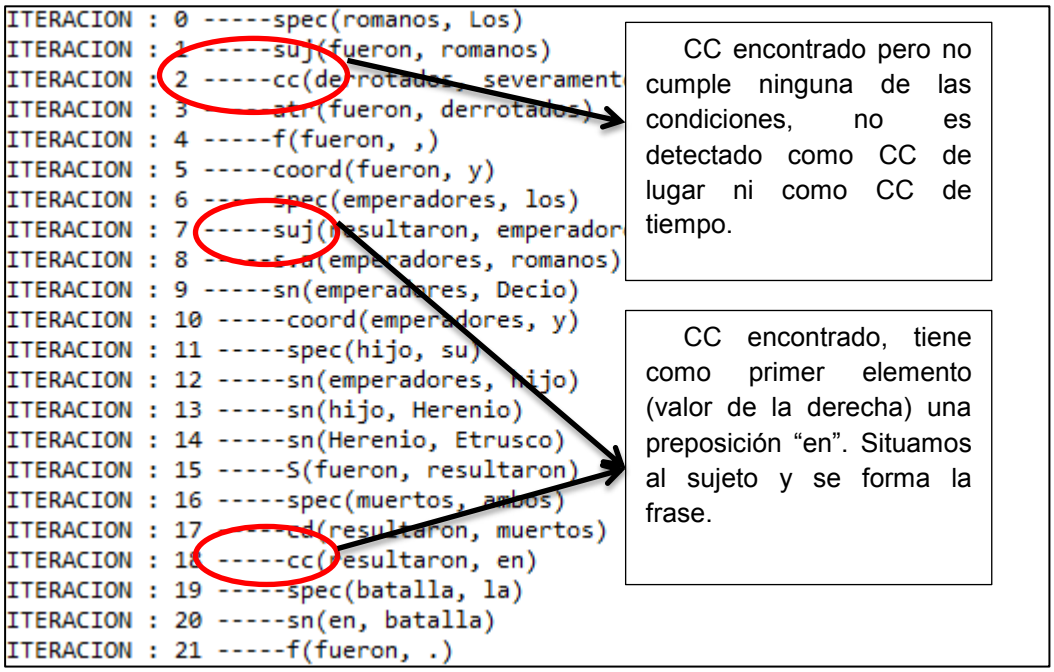


Figura 5-34: Salida del sistema, generación de la pregunta Dónde.

Pregunta generada: ***¿Dónde resultaron muertos los emperadores romanos Decio y su hijo Herenio Etrusco?***

- Generación de la pregunta Cuándo.

  - Se empezará la pregunta con el símbolo de interrogación "¿" seguido de la palabra "Cuándo".
  - Se deberá situar el sujeto mediante el procedimiento situarSujeto, en este paso se rellenará la variable global palabraP.
  - Se deberá formar la frase mediante la función formarFrase, se devolverá la frase la cual llamaremos fraseP.
  - Se añadirá a la pregunta palabraP seguida de fraseP.
  - Se terminará añadiendo el símbolo de interrogación "?".

Ejemplo:

1- *La batalla de Anquialo tuvo lugar en el año 763, cerca de la ciudad de Pomorie en la costa búlgara del mar Negro.*

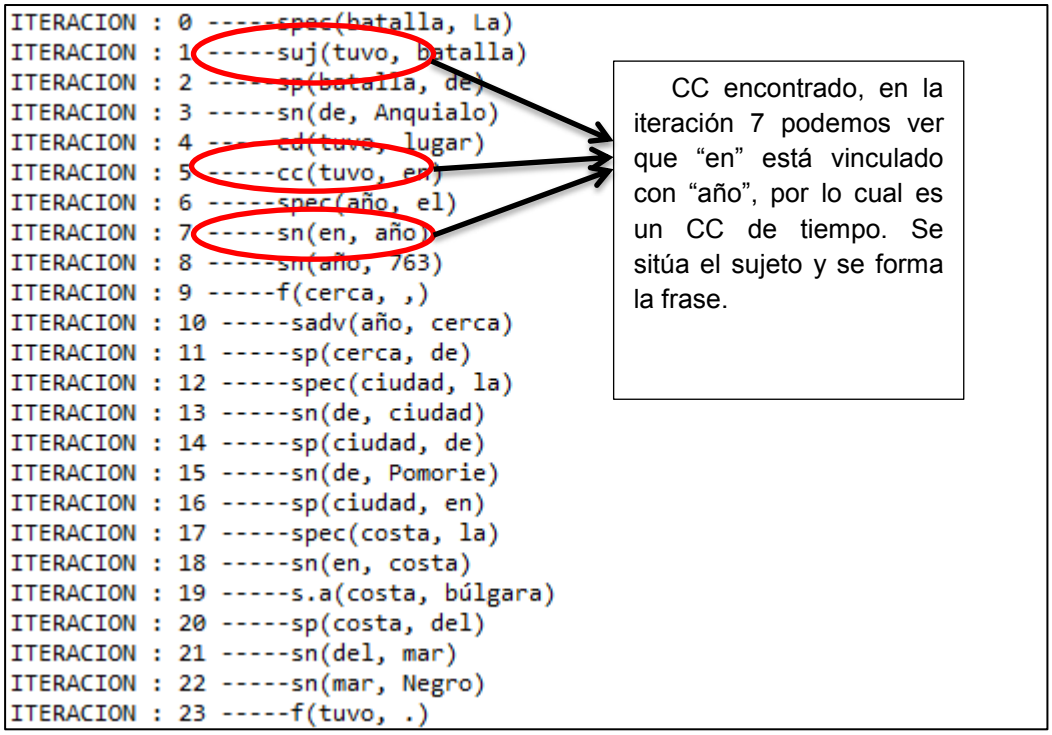


Figura 5-35: Salida del sistema, generación de la pregunta Cuándo.

Pregunta generada: ***¿Cuándo tuvo lugar la batalla de Anquialo?***

Capítulo 6

Evaluación

En este capítulo se expondrán las diversas evaluaciones realizadas a lo largo del proyecto para darle una mayor credibilidad y para demostrar si es un generador eficiente o que tan eficiente puede llegar a ser.

Estará dividido en dos sub apartados:

- El primero tratará sobre las evaluaciones que respaldan los algoritmos creados a partir de un generador de preguntas basado en un analizador morfosintáctico y en un analizador basado en dependencias sintácticas.

- El segundo será una evaluación realizada por dos personas ajenas al proyecto, estas evaluación serán medidas mediante el coeficiente kappa de cohen (18).

El Coeficiente kappa de Cohen mide el acuerdo entre dos observadores en sus correspondientes clasificaciones de N elementos en C categorías mutuamente excluyentes.

La ecuación para κ es:

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)},$$

Donde Pr (a) es el acuerdo observado relativo entre los observadores, y Pr (e) es la probabilidad hipotética de acuerdo por azar, utilizando los datos observados para calcular las probabilidades de que cada observador clasifique aleatoriamente cada categoría. Si los evaluadores están completamente de acuerdo, entonces κ = 1. Si no hay acuerdo entre los calificadores distinto al que cabría esperar por azar (según lo definido por Pr (e)), κ = 0.

Para calcular el coeficiente de kappa se usó la página web: statstodo (19).

6.1. Evaluación de los algoritmos

Este tipo de evaluación fue realizada con el propósito de dar mayor confianza individual a los algoritmos desarrollados. Todas tienen el mismo procedimiento para evaluarlas.

6.1.1. Evaluación de los algoritmos de un generador de preguntas a partir de un análisis morfosintáctico.

- **Evaluación del algoritmo Qué.**

  Para esta evaluación se seleccionaron de manera manual 50 sentencias de un total de 20 textos, en las cuales con cada una de ellas se podía formar una pregunta del tipo Qué.

  Cada una de estas sentencias fue analizada en base al algoritmo desarrollado para ver si se detectaba el tipo de pregunta.

  Como resultado dio que de estas 50 sentencias, 43 fueron detectadas correctamente.

  Esto supone que el 86% fueron acertadas.

  A continuación podemos ver un cuadro con 4 de estas sentencias las cuales tuvieron un resultado positivo.

Sentencias		Análisis rápido	Pregunta		Pregunta detectada con el algoritmo
La misión del papa llegó a Bulgaria a finales de verano o durante el otoño de 926, llevando una corona y el cetro con el que se coronaria a Simeón como emperador de Bulgaria.		SN La misión del papa GRUP.VERB llegó SP a Bulgaria SP a finales de verano o durante el otoño de 926 S , llevando una corona y el cetro con el que se coronaria a Simeón como emperador de Bulgaria FP .	¿Qué llegó a Bulgaria a finales de verano o durante el otoño de 926, llevando una corona y el cetro con el que se coronaria a Simeón como emperador de Bulgaria?		Qué
La batalla de Mursa Major tuvo lugar en 351 entre un ejército romano dirigido por el emperador Constancio II y las fuerzas del usurpador Magnencio.		SN La batalla de Mursa SN Major GRUP.VERB tuvo SN lugar en 351 entre un ejército romano dirigido por el emperador Constancio II y las fuerzas del usurpador Magnencio FP .	¿Qué tuvo lugar en 351 entre un ejército romano dirigido por el emperador Constancio II y las fuerzas del usurpador Magnencio?		Qué
La influencia de los Francos llegó a su fin		Tipo: SN Texto: la influencia de los Francos Tipo: GRUP.VERB Texto: llegó Tipo: SP Texto: a su fin	¿Qué llegó a su fin?		Qué
La batalla acabó sin muchas bajas.		SN la batalla GRUP.VERB acabó SP sin muchas bajas FP.	¿Qué acabó sin muchas bajas?		Qué

Tabla 6-1: Estudio del algoritmo Qué.

- **Evaluación del algoritmo Quién/Quiénes.**

  Para esta evaluación se seleccionaron de manera manual 50 sentencias de un total de 20 textos, en las cuales con cada una de ellas se podía formar una pregunta del tipo Quién/Quiénes.

  Cada una de estas sentencias fue analizada en base al algoritmo desarrollado para ver si se detectaba el tipo de pregunta.

  Como resultado dio que de estas 50 sentencias, 41 fueron detectadas correctamente.

  Esto supone que el 82% fueron acertadas.

  A continuación podemos ver un cuadro con 4 de estas sentencias las cuales tuvieron un resultado positivo.

Sentencias		Análisis rápido	Pregunta		Pregunta detectada con el algoritmo
El gobernante búlgaro, Simeón, era un hombre sabio y capaz, con un espíritu inquieto e insaciable.		SN El gobernante búlgaro , Simeón , GRUP.VERB era SN un hombre sabio y capaz , con un espíritu inquieto e insaciable FP .	¿Quién era un hombre sabio y capaz, con un espíritu inquieto e insaciable?		Quién
Los búlgaros fueron recibidos por el ejército de Tomislav en la región montañosa del este de Bosnia el 27 de mayo de 927.		SN Los búlgaros GRUP.VERB fueron recibidos SP por el ejército de Tomislav SP en la región montañosa del este de Bosnia SN el 27 de mayo de 927 FP .	¿Quiénes fueron recibidos por el ejército de Tomislav en la región montañosa del este de Bosnia el 27 de mayo de 927?		Quiénes
El comandante de las fuerzas búlgaras en esta batalla fue el Duque Alogobotur.		SN El comandante de las fuerzas búlgaras en esta batalla GRUP.VERB fue SN el Duque Alogobotur FP .	¿Quién fue el Duque Alogobotur?		Quién
Constantino y su ejército estaban persiguiendo algunos sármatas que habían cruzado el río Danubio al territorio de Licinio.		SN Constantino y su ejército GRUP.VERB estaban persiguiendo SN algunos sármatas que habían cruzado el río Danubio al territorio de Licinio FP .	¿Quiénes estaban persiguiendo algunos sármatas que habían cruzado el río Danubio al territorio de Licinio?		Quiénes

Tabla 6-2: Estudio del algoritmo Quién/Quiénes.

- **Evaluación del algoritmo Cuándo**

  Para esta evaluación se seleccionaron de manera manual 50 sentencias de un total de 20 textos, en las cuales con cada una de ellas se podía formar una pregunta del tipo Cuándo.

  Cada una de estas sentencias fue analizada en base al algoritmo desarrollado para ver si se detectaba el tipo de pregunta.

  Como resultado dio que de estas 50 sentencias, 33 fueron detectadas correctamente.

  Esto supone que el 66% fueron acertadas.

  A continuación podemos ver un cuadro con 4 de estas sentencias las cuales tuvieron un resultado positivo.

Sentencias		Análisis rápido	Pregunta		Pregunta detectada con el algoritmo
A su vez , Saladino unió bajo su mando los ejércitos de Siria y Egipto , y tras un breve e infructuoso asedio de Tiro , el sultán llegó a las afueras de Jerusalén el 20 de septiembre.		SP A su vez S , Saladino unió bajo su mando los ejércitos de Siria y Egipto , y tras un breve e infructuoso asedio de Tiro , SN el sultán GRUP.VERB llegó SP a las afueras de Jerusalén SN el 20 de septiembre FP .	¿Cuándo llegó a las afueras de Jerusalén el sultán?		Cuándo
La batalla tuvo lugar el 27 de mayo de 927.		SN La batalla GRUP.VERB tuvo SN lugar SN el 27 de mayo de 927 FP .	¿Cuándo tuvo lugar la batalla?		Cuándo
El Combate de los Treinta fue una acción militar acaecida el 27 de marzo de 1351.		SN El Combate de los Treinta GRUP.VERB fue SN una acción militar acaecida el 27 de marzo de 1351 FP .	¿Cuándo fue El Combate de los Treinta?		Cuándo
La batalla de Domažlice o batalla de Taus fue un enfrentamiento armado que se libró el 14 de agosto.		SN La batalla de Domažlice o batalla de Taus GRUP.VERB fue SN un enfrentamiento armado que se libró el 14 de agosto. FP .	¿Cuándo fue la batalla de Domažlice o batalla de Taus?		Cuándo

Tabla 6-3: Estudio del algoritmo Cuándo.

6.1.2. Evaluación de los algoritmos de un generador de preguntas a partir de un análisis basado en dependencias sintácticas.

- **Evaluación del algoritmo Cuándo**

Para realizar esta evaluación se tomaron las mismas sentencias que se usaron en la evaluación del algoritmo Cuándo del generador basado en un análisis morfosintáctico.

Cada una de estas sentencias fue analizada en base al algoritmo desarrollado para ver si se detectaba el tipo de pregunta.

Como resultado dio que de estas 50 sentencias, 3 fueron detectadas correctamente.

Esto supone que el 6% fueron acertadas.

Esto se debe a que las fechas por lo general no eran detectadas como complemento circunstancial.

- **Evaluación del algoritmo Dónde**

Para realizar esta evaluación se tomaron las mismas sentencias que se usaron en la evaluación del algoritmo Dónde del generador basado en un análisis morfosintáctico.

Cada una de estas sentencias fue analizada en base al algoritmo desarrollado para ver si se detectaba el tipo de pregunta.

Como resultado dio que de estas 50 sentencias, 30 fueron detectadas correctamente.

Esto supone que el 60% fueron acertadas.

A continuación podemos ver un cuadro con 2 de estas sentencias las cuales tuvieron un resultado positivo.

Sentencias	Análisis rápido	Pregunta	Pregunta detectada con el algoritmo
Ambos emperadores fueron elegidos y apoyados por el Senado Romano y su asentamiento se localizaba en la provincia de África.	ITERACION : 0 -----spec(emperadores, Ambos) ITERACION : 1 -----suj(elegidos, emperadores) ITERACION : 2 -----v(elegidos, fueron) ITERACION : 3 -----coord(elegidos, y) ITERACION : 4 -----S(elegidos, apoyados) ITERACION : 5 -----cag(elegidos, por) ITERACION : 6 -----spec(Senado, el) ITERACION : 7 -----sn(por, Senado) ITERACION : 8 -----sn(Senado, Romano) ITERACION : 9 -----coord(elegidos, y) ITERACION : 10 -----spec(asentamiento, su) ITERACION : 11 -----suj(localizaba, asentamiento) ITERACION : 12 -----morfema.pronominal(localizaba, se) ITERACION : 13 -----S(elegidos, localizaba) ITERACION : 14 -----cc(localizaba, en) ITERACION : 15 -----spec(provincia, la) ITERACION : 16 -----sn(en, provincia) ITERACION : 17 -----sp(provincia, de) ITERACION : 18 -----sn(de, Africa) ITERACION : 19 -----f(elegidos, .)	¿Dónde se localizaba su asentamiento?	Dónde
El resto de los muertos se reunieron en varias fosas comunes.	ITERACION : 0 -----spec(resto, El) ITERACION : 1 -----suj(reunieron, resto) ITERACION : 2 -----sp(resto, de) ITERACION : 3 -----spec(muertos, los) ITERACION : 4 -----sn(de, muertos) ITERACION : 5 -----morfema.pronominal(reunieron, se) ITERACION : 6 -----cc(reunieron, en) ITERACION : 7 -----spec(fosas, varias) ITERACION : 8 -----sn(en, fosas) ITERACION : 9 -----s.a(fosas, comunes) ITERACION : 10 -----f(reunieron, .)	¿Dónde reunieron El resto de los muertos?	Dónde

Tabla 6-4: Estudio del algoritmo Dónde.

6.2. **Evaluación del generador**

Las siguientes evaluaciones han sido desarrolladas por personas ajenas al proyecto con el objetivo de darle una mayor credibilidad.

6.2.1. Evaluación del generador de preguntas a partir de un analizador morfosintáctico.

Los evaluadores de este generador fueron:

- Evaluador 1 - Anartz Recalde
- Evaluador 2 - Jose Alfaro

Para llevar a cabo la evaluación se creó dos ficheros, estos ficheros están vinculados el uno con el otro ya que la información de cada fila de ambos fichero está relacionada de manera directa, mientras que en el Fichero A en la columna 1 se encuentran las sentencias, en el Fichero B en la columna 2 se encuentran las mismas sentencias, mientras que en el Fichero A en la columna 3 se encuentra el pronombre interrogativo (tipo de pregunta), en el Fichero B en la columna 1 se encuentra la pregunta generada usando ese pronombre interrogativo. Ambos evaluadores evaluaron los dos ficheros.

Los ficheros tienen las siguientes características:

- Fichero A: es un fichero que está dividido por 4 columnas.

  - Columna 1: En cada fila de esta columna hay una **sentencia**.
  - Columna 2: En cada fila de esta columna está el **pronombre interrogativo** (tipo de pregunta) que los distintos algoritmos han detectado vinculados a las sentencias de la columna 1.
  - Columna 3: En cada fila de esta columna se da la frase que **respondería** a la pregunta.
  - Columna 4: En cada fila de esta columna se deberá evaluar si el **pronombre interrogativo es correcto**, para ello se pondrá un 1 si es correcto o un 0 si es incorrecto. Luego se deberá ver si mediante el pronombre interrogativo habría alguna manera de formar alguna pregunta en la sentencia de la columna 1 y que esta tenga coherencia con la respuesta de la columna 3.

  A continuación se muestra un cuadro como ejemplo con las características descritas:

Frase Original	Pronombre Interrogativo	Respuesta	¿Pronombre interrogativo correcto?
El gobernante búlgaro, Simeón , era un hombre sabio y capaz , con un espíritu inquieto e insaciable .	Quién	El gobernante búlgaro , Simeón ,	1
Su objetivo fundamental era la derrota del Imperio bizantino y conquistar Bizancio.	Quién	Su objetivo fundamental	0
Para lograr su objetivo , Simeón invadió el oriente y centro de los Balcanes en varias ocasiones , ocupando Serbia y finalmente atacando a Croacia .	Quién	Simeón	1
Después de largas guerras y grandes éxitos, capturando la mayor parte del territorio bizantino en Europa , Simeón el Grande fue coronado en la iglesia de Ohrid como " zar de todos los búlgaros y los griegos " por el recién nombrado patriarca búlgaro en 925 .	Quién	Simeón el Grande	1
De acuerdo con el razonamiento jurídico de la época, sólo el papa y el emperador bizantino podían otorgar títulos reales o imperiales, y un emperador podría ser coronado sólo por un patriarca .	Quiénes	sólo el papa y el emperador bizantino	1
De acuerdo con el razonamiento jurídico de la época , sólo el papa y el emperador bizantino podían otorgar títulos reales o imperiales , y un emperador podría ser coronado sólo por un patriarca .	Qué	un emperador	0

Tabla 6-5: Fichero A de la evaluación del generador a partir de un análisis morfosintáctico.

- Fichero B: Está dividido en 4 columnas:

  - Columna 1: En cada fila de esta columna se encuentra **la pregunta** generada por el programa.
  - Columna 2: En cada fila de esta columna se encuentra **la sentencia** a partir de la cual se ha generado la pregunta.
  - Columna 3: En cada fila de esta columna se evalúa si la pregunta es **correcta sintácticamente** (si tiene sentido), su evaluación es de un 1 si es correcta o un 0 si es incorrecta.
  - Columna 4. En cada fila de esta columna se evalúa si la **pregunta es adecuada**. La pregunta será adecuada si aparte de ser correcta sintácticamente es relevante con respecto a la sentencia.

A continuación se muestra un cuadro como ejemplo con las características descritas:

Preguntas	Frase Original	¿Correcta sintácticamente?	¿Es una pregunta adecuada?
¿Quién era un hombre sabio y capaz , con un espíritu inquieto e insaciable .?	El gobernante búlgaro , Simeón , era un hombre sabio y capaz , con un espíritu inquieto e insaciable .	1	1
¿Quién era la derrota del Imperio bizantino y conquistar Bizancio .?	Su objetivo fundamental era la derrota del Imperio bizantino y conquistar Bizancio .	0	0
¿Quién invadió el oriente y centro de los Balcanes en varias ocasiones , ocupando Serbia y finalmente atacando a Croacia .?	Para lograr su objetivo , Simeón invadió el oriente y centro de los Balcanes en varias ocasiones , ocupando Serbia y finalmente atacando a Croacia .	1	1
¿Quién fue coronado en la iglesia de Ohrid como " zar de todos los búlgaros y los griegos " por el recién nombrado patriarca búlgaro en 925 .?	Después de largas guerras y grandes éxitos , capturando la mayor parte del territorio bizantino en Europa , Simeón el Grande fue coronado en la iglesia de Ohrid como " zar de todos los búlgaros y los griegos " por el recién nombrado patriarca búlgaro en 925 .	1	1
¿Quiénes podían otorgar títulos reales o imperiales ,?	De acuerdo con el razonamiento jurídico de la época , sólo el papa y el emperador bizantino podían otorgar títulos reales o imperiales , y un emperador podría ser coronado sólo por un patriarca .	1	1
¿Qué podría ser coronado sólo por un patriarca?	De acuerdo con el razonamiento jurídico de la época , sólo el papa y el emperador bizantino podían otorgar títulos reales o imperiales , y un emperador podría ser coronado sólo por un patriarca .	0	0

Tabla 6-6: Fichero B de la evaluación del generador a partir de un análisis morfosintáctico.

Estos ficheros se crearon de manera automática mediante un programa que se creó especialmente para evaluarlos, en el **¡Error! No se encuentra el origen de la referencia.** se encuentra el código para la creación de estos ficheros.

Hay que destacar que solo se evaluaron dos tipos de preguntas en estos ficheros, las preguntas del tipo Qué y las del tipo Quién/Quiénes.

Estas evaluaciones se repitieron dos veces en distintas fechas. A continuación se dan los resultados cada una de las evaluaciones.

- ## Evaluación A

Fecha de la evaluación: 07/12/15.
Textos pertenecientes al training.
Número de textos usados: 28.
Número de sentencias: 306.
Número de preguntas: 155
Número de preguntas de tipo Quién: 61.
Número de preguntas de tipo Quiénes: 15.
Número de preguntas de tipo Qué: 79.

A continuación se muestran el grado de acuerdo según el coeficiente kappa entre los dos evaluadores.

- En relación al fichero A: ¿Es un pronombre interrogativo correcto?

Count matrix

	0	1	Tot
0	29	4	33
1	14	108	122
Tot	43	112	155

Cohen's Kappa Unweighted = 0.6880 SE = 0.0691
95%CI = 0.5525 to 0.8235

Cohen's Kappa Weighted = 0.6880 SE = 0.0691
95%CI = 0.5525 to 0.8235

Según el resultado, el acuerdo entre los dos evaluadores es de un 0,6880 o lo que es lo mismo a un 68,8%.

- En relación al fichero B ¿Es correcta sintácticamente?

Count matrix

	0	1	Tot
0	29	6	35
1	5	115	120
Tot	34	121	155

Cohen's Kappa Unweighted = 0.7949 SE = 0.0596
95%CI = 0.6782 to 0.9117

Cohen's Kappa Weighted = 0.7949 SE = 0.0596
95%CI = 0.6782 to 0.9117

Según el resultado, el acuerdo entre los dos evaluadores es de un 0,7949 o lo que es lo mismo a un 79,49%.

- En relación al fichero B: ¿Es una pregunta adecuada?

Count matrix

	0	1	Tot
0	31	19	50
1	13	92	105
Tot	44	111	155

Cohen's Kappa Unweighted = 0.5123 SE = 0.0768
95%CI = 0.3618 to 0.6628

Cohen's Kappa Weighted = 0.5123 SE = 0.0768
95%CI = 0.3618 to 0.6628

Según el resultado, el acuerdo entre los dos evaluadores es de un 0,5123 o lo que es lo mismo a un 51,23%.

Análisis de Resultados

El siguiente cuadro refleja el número de preguntas correctas respecto a su tipo de pregunta.

Evaluador	¿Pronombre interrogativo correcto?			¿Correcta sintácticamente?			¿Pregunta adecuada?		
	Qué	Quién /Quiénes	Total	Qué	Quién /Quiénes	Total	Qué	Quién /Quiénes	Total
Evaluador 1	54	58	112	55	66	121	46	65	111
Evaluador 2	62	60	122	54	66	120	46	59	105

Tabla 6-7: Preguntas correctas respecto a su tipo de pregunta.

Teniendo en cuenta que hay un total de 155 preguntas, en los siguientes cuadros se realiza una comparación entre los resultados individuales de cada evaluador y el resultado según el grado de acuerdo del coeficiente Kappa de cohen.

¿Pronombre interrogativo correcto?			
	Porcentaje del total respecto a las 155 preguntas	Porcentaje del Coeficiente kappa de cohen	Diferencia
Evaluador 1	72,25%	68,8%.	3,45%
Evaluador 2	78,70%	68,8%.	9,9%

Tabla 6-8: Comparación pronombre interrogativo

Como podemos observar ambos resultados individuales se ajustan muy bien al grado de acuerdo del coeficiente Kappa, teniendo en ambos casos una diferencia que no llega a sobrepasar el 10%.

¿Correcta sintácticamente?			
	Porcentaje del total respecto a las 155 preguntas	Porcentaje del Coeficiente kappa de cohen	Diferencia
Evaluador 1	78,06%	79,49%.	1,43%
Evaluador 2	77,41%	79,49%.	2,08%

Tabla 6-9: Comparación de la sintáctica

En esta tabla se muestra unos resultados bastantes favorables. Como se puede observar ambos evaluadores están de acuerdo en un grado bastante alto, en ambos casos se puede ver que la diferencia es mínima, oscila alrededor del 2%.

¿Pregunta adecuada?			
	Porcentaje del total respecto a las 155 preguntas	Porcentaje del Coeficiente kappa de cohen	Diferencia
Evaluador 1	71,61%	51,23%	20,38%
Evaluador 2	67,74%	51,23%	16,53%

Tabla 6-10: Comparación sobre si es una pregunta adecuada.

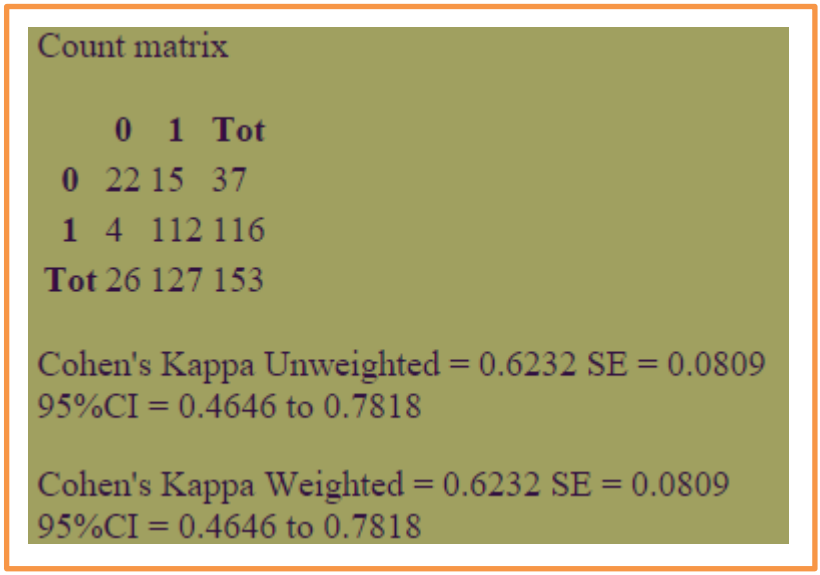
En este último cuadro sin embargo el grado de acuerdo entre ambos evaluadores es muy bajo, habiendo una gran diferencia entre el resultado individual y el del coeficiente Kappa.

Como apunte final de esta evaluación podemos decir que la identificación del pronombre interrogativo y la sintaxis de la pregunta formada tienen buenos resultados, mientras que la relevancia de la pregunta con respecto a la sentencia debería mejorarse.

- **Evaluación B**

Fecha de la evaluación: 25/01/16.
Textos pertenecientes a la evaluación.
Número de textos usados: 34.
Número de sentencias: 382.
Número de preguntas: 153.
Número de preguntas de tipo Quién: 49.
Número de preguntas de tipo Quiénes: 27.
Número de preguntas de tipo Qué: 77.

- En relación al fichero A: ¿Es un pronombre interrogativo correcto?



Count matrix

	0	1	Tot
0	22	15	37
1	4	112	116
Tot	26	127	153

Cohen's Kappa Unweighted = 0.6232 SE = 0.0809
95%CI = 0.4646 to 0.7818

Cohen's Kappa Weighted = 0.6232 SE = 0.0809
95%CI = 0.4646 to 0.7818

Según el resultado, el acuerdo entre los dos evaluadores es de un 0,6232 o lo que es lo mismo a un 62,32%.

- En relación al fichero B ¿Es correcta sintácticamente?

Count matrix

	0	1	Tot
0	12	10	22
1	1	130	131
Tot	13	140	153

Cohen's Kappa Unweighted = 0.6481 SE = 0.1022
95%CI = 0.4478 to 0.8485

Cohen's Kappa Weighted = 0.6481 SE = 0.1022
95%CI = 0.4478 to 0.8485

Según el resultado, el acuerdo entre los dos evaluadores es de un 0,6481 o lo que es lo mismo a un 64,81%.

- En relación al fichero B: ¿Es una pregunta adecuada?

Count matrix

	0	1	Tot
0	27	17	44
1	12	97	109
Tot	39	114	153

Cohen's Kappa Unweighted = 0.5212 SE = 0.0800
95%CI = 0.3643 to 0.6781

Cohen's Kappa Weighted = 0.5212 SE = 0.0800
95%CI = 0.3643 to 0.6781

Según el resultado, el acuerdo entre los dos evaluadores es de un 0,5212 o lo que es lo mismo a un 52,12%.

Análisis de Resultados

El siguiente cuadro refleja el número de preguntas correctas respecto a su tipo de pregunta.

Evaluador	¿Pronombre interrogativo correcto?			¿Correcta sintácticamente?			¿Pregunta adecuada?		
	Qué	Quién /Quiénes	Total	Qué	Quién /Quiénes	Total	Qué	Quién /Quiénes	Total
Evaluador 1	56	56	112	46	64	110	43	64	107
Evaluador 2	68	59	127	75	72	147	59	55	114

Tabla 6-11: Preguntas correctas respecto a su tipo de pregunta.

Teniendo en cuenta que hay un total de 153 preguntas, en los siguientes cuadros se realiza una comparación entre los resultados individuales de cada evaluador y el resultado según el grado de acuerdo del coeficiente Kappa de cohen.

¿Pronombre interrogativo correcto?			
	Porcentaje del total respecto a las 153 preguntas	Porcentaje del Coeficiente kappa de cohen	Diferencia
Evaluador 1	73,20%	62,32%	10,88%
Evaluador 2	83,03%	62,32%	20,71%

Tabla 6-12: Comparación pronombre interrogativo

El grado de acuerdo según el porcentaje del Coeficiente kappa de cohen entre ambos evaluadores es del 62,32%, pero si observamos que tanta diferencia hay individualmente podemos observar que el evaluador 1 tiene un 10,88%, mientras que el evaluador 2 tiene un 20,71%.

¿Correcta sintácticamente?			
	Porcentaje del total respecto a las 153 preguntas	Porcentaje del Coeficiente kappa de cohen	Diferencia
Evaluador 1	71,89%	64,81%	7,08%
Evaluador 2	96,07%	64,81%	31,26%

Tabla 6-13: Comparación de la sintáctica

En esta tabla se muestra unos resultados los cuales indican que el grado de acuerdo según el porcentaje del Coeficiente kappa entre ambos evaluadores es del 64,81%, aunque cabe destacar que el evaluador 2 ha tenido una diferencia del 31,26%.

¿Pregunta adecuada?			
	Porcentaje del total respecto a las 153 preguntas	Porcentaje del Coeficiente kappa de cohen	Diferencia
Evaluador 1	69,93%	52,12%	17,81%
Evaluador 2	74,5%	52,12%	22,38%

Tabla 6-14: Comparación sobre si es una pregunta adecuada.

En este último cuadro se puede observar que el grado de acuerdo entre ambos evaluadores es del 52,12%, habiendo una gran diferencia entre el resultado individual y el del coeficiente Kappa.

Como apunte final de esta evaluación podemos decir que la identificación del pronombre interrogativo y la sintaxis de la pregunta formada tienen unos resultados aceptables, mientras que la relevancia de la pregunta con respecto a la sentencia, al igual que en la anterior evaluación, debería mejorarse.

6.2.2. Evaluación del generador de preguntas a partir de un analizador basado en dependencias sintácticas.

Evaluador:

- José Alfaro

En la siguiente evaluación sólo se evaluaron preguntas del tipo Dónde, ya que la evaluación del algoritmo Cuándo fue insatisfactoria, sólo se repitió una vez y en esta ocasión solo se contó con un evaluador. Para esta evaluación se crearon los ficheros A y B descritos en el anterior punto.

Fecha de la evaluación: 30/01/16.
Textos pertenecientes al Test.
Número de textos usados: 7.
Número de sentencias: 146.
Número de preguntas: 130.
Número de preguntas de tipo Dónde

Al no haber dos evaluadores no se usa el Coeficiente kappa de Cohen como medidor de acuerdo, simplemente se saca el porcentaje de aciertos que ha habido en cada pregunta.

- En relación al fichero A: ¿Es un pronombre interrogativo correcto?

  El evaluador indicó que de las 130 preguntas, 75 le parecieron que eran acertadas, lo que supone un porcentaje del 57,60%.

- En relación al fichero B: ¿Es correcta sintácticamente?

  El evaluador indicó que de las 130 preguntas, 70 le parecieron que eran acertadas, lo que supone un porcentaje del 53,83%.

- En relación al fichero B: ¿Es una pregunta adecuada?

El evaluador indicó que de las 130 preguntas, 60 le parecieron que eran acertadas, lo que supone un porcentaje del 46,15%.

Capítulo 7

Gestión del proyecto

En este capítulo se presenta todo lo relacionado con la gestión del proyecto. Para ello se muestran los siguientes apartados:

- El EDT (estructura de descomposición del trabajo).
- Diagrama de Gantt.
- La gestión de costes
- La gestión del tiempo.

7.1. **EDT**

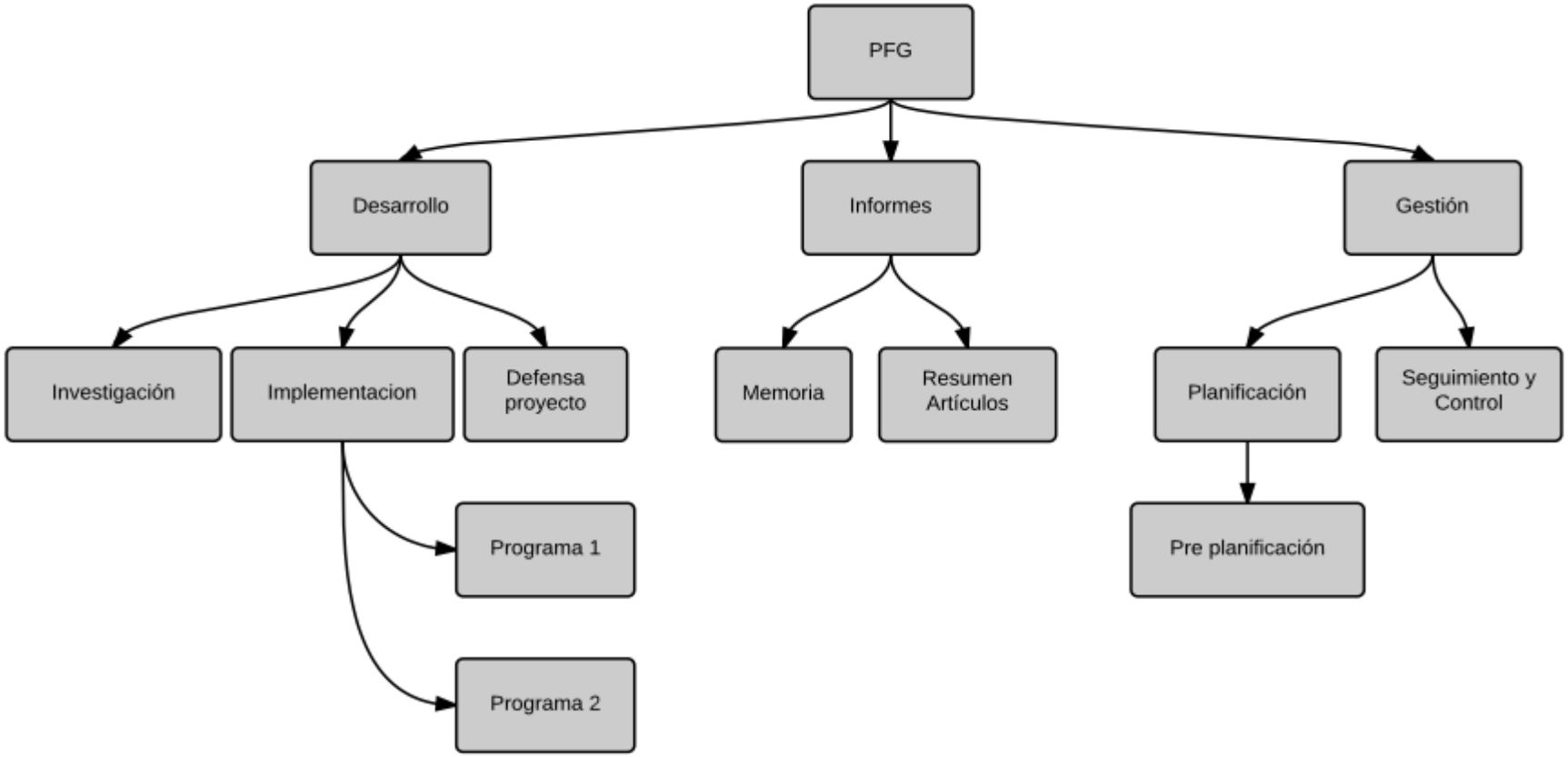


Figura 7-1: EDT

7.2. **Diagrama de Gantt**

	MAR		ABR		MAY		SEP		OCT		NOV		DIC		ENE		FEB	
Inicio del proyecto																		
Fin del proyecto																		
1. Desarrollo																		
1.1. Investigación																		
1.1.1. Lectura de artículos																		
1.1.2. Herramientas IXA-Pipes																		
1.1.3. Obtención del corpus																		
1.1.4. Definición de la arquitectura																		
1.1.5. Algoritmos de generación de preg																		
1.2 Implementación																		
1.2.1. Programa 1 Gen. de Pre. Morf																		
1.2.2. Programa 2 Gen. de Pre. Basado en Dep																		
2. Informes																		
2.1 Memoria																		
2.2 Resumen de Artículos																		
3. Gestión																		
3.1 planificación																		
3.2 Seguimiento y control																		

Tabla 7-1: Diagrama de Gantt

	Parte común
	1era fase
	2da fase

7.3. **Gestión de los costes**

El coste de este proyecto ha sido únicamente las horas invertidas en llevar a cabo su realización. En la siguiente tabla se muestran la dedicación medidas en horas.

Categoría	Tarea	Dedicación (horas)
Investigación	Lectura y síntesis de los artículos	30
	Herramientas Ixa-Pipes	40
	Obtención del corpus	20
	Definición de la arquitectura	20
	Algoritmos de la Generación de Preguntas	45
Implementación	Programa 1: Generador de preguntas a partir de un análisis morfosintáctico.	110
	Programa 2: Generador de preguntas a partir de un análisis basado en dependencias.	60
Otros	Redacción de la memoria	70
	Reuniones de seguimiento y control	10
	Planificación	10
	Preparación de la defensa (estimado)	15
Total		**430**

Tabla 7-2: Dedicación del proyecto

7.4. **Gestión del tiempo**

Este proyecto dio comienzo el día 9 de marzo del 2015 y finalizó el 15 de febrero con la defensa del proyecto. Durante este transcurso hubo fechas que se han identificado como las más relevantes. A continuación se presentan los hitos más importantes:

- 09/03/15, inicio del proyecto.
- 10/11/15, finalización de la implementación del primer generador.

- 26/11/16, finalización de la implementación del segundo generador.
- 10/12/15, resultados de la evaluación del primer generador
- 30/11/16, resultados de la evaluación del segundo generador.
- 03/11/15, primera versión de la memoria.
- 21/01/16, segunda versión de la memoria.
- 31/01/16, versión final de la memoria.

También cabe destacar que a lo largo del proyecto ha habido un total 21 reuniones con la directora del proyecto, generalmente se hacía una reunión por semana.

Capítulo 8

Conclusiones

Como capítulo final se presentan las conclusiones, estas están divididas en tres apartados. El primero trata sobre las conclusiones generales del proyecto. El segundo es un apartado que hace una pequeña introducción al Mapa conceptual y como se podría generar preguntas a partir de él. El último apartado son mejoras que se podrían hacer sobre este proyecto.

8.1. Conclusiones generales del proyecto

El proyecto desarrollado ha conseguido sus principales objetivos, crear dos aplicaciones capaces de generar preguntas. Es cierto que se ha invertido más tiempo en el generador de preguntas a partir de un análisis morfosintáctico, en la evaluación se refleja, pero el generador de preguntas a partir de un análisis basado en dependencias también tiene unos resultados satisfactorios. Algo bueno que tienen ambos generadores es que pueden ser mejorados continuamente, solo hay que depurar los algoritmos.

También se han conseguido los objetivos secundarios, se han creado diversos algoritmos a lo largo de este proyecto al mismo tiempo que se aprendía a usar diversas herramientas del PLN como son las herramientas lxa-pipes. La mayoría de estos algoritmos creados han dado resultados positivos.

Ha sido de gran ayuda cursar asignaturas como diseño de algoritmos, en diversos puntos del proyecto se han usado técnicas aprendidas de esta asignatura, así como minería de datos e inteligencia artificial, ambas muy vinculadas al PLN.

Se puede identificar el inicio del proyecto como la parte más difícil del mismo, ya que la parte de la investigación fue un descubrimiento de nuevos términos, de nuevas herramientas, de nuevas técnicas, de nuevos autores, etc…, todos estos nuevos conceptos que se apilaban uno detrás de otro al inicio del proyecto no dejaban ver con claridad el alcance, la magnitud que conllevaba elaborar un proyecto de este calibre. Fue por eso que se optó por realizar una pre-planificación, haber cursado gestión de proyectos ha sido de gran ayuda a la hora de la gestión, de esta forma primero se investigó hasta donde se podría llegar y luego se definieron el alcance y los objetivos.

En general ha sido un proyecto enriquecedor, con un producto final que puede ser de interés en futuros proyectos.

8.2. Mapa conceptual

Como ya se comentó en el apartado 1.1, al inicio del proyecto se pensó en la posibilidad de generar preguntas a partir de un mapa conceptual, para ello se elaboraron distintos diseños y se estudiaron diferentes aspectos del mapa conceptual, en este apartado se mostrará todo lo relacionado a lo estudiado ya que de alguna forma puede servir como referencia para posibles trabajos vinculados con este proyecto.

8.2.1. Definición de Mapa Conceptual

Es una técnica usada para representar gráficamente el conocimiento, por ello se dice que es una red de conceptos. En esta red los nodos representan los conceptos, y los enlaces (aristas) representan las relaciones entre estos conceptos

Un mapa conceptual se puede caracterizar porque:

- Su estructura es jerárquica.
- Su representación debe de poder darse de manera gráfica mediante figuras geométricas.

A continuación se propone un ejemplo genérico que no está vinculado con la generación de preguntas, es más que todo para tener una idea general de en qué consiste un mapa conceptual, más adelante se realizarán ejemplos vinculados a la generación de preguntas.

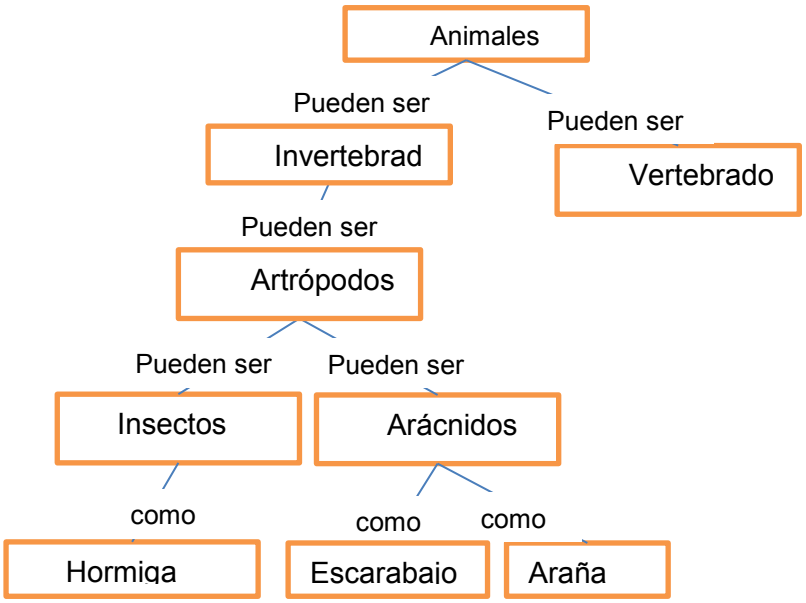


Figura 8-1: Mapa conceptual animal

8.2.2. Concept Maps Editor

Es un software de edición que permite la creación de mapas conceptuales (20).

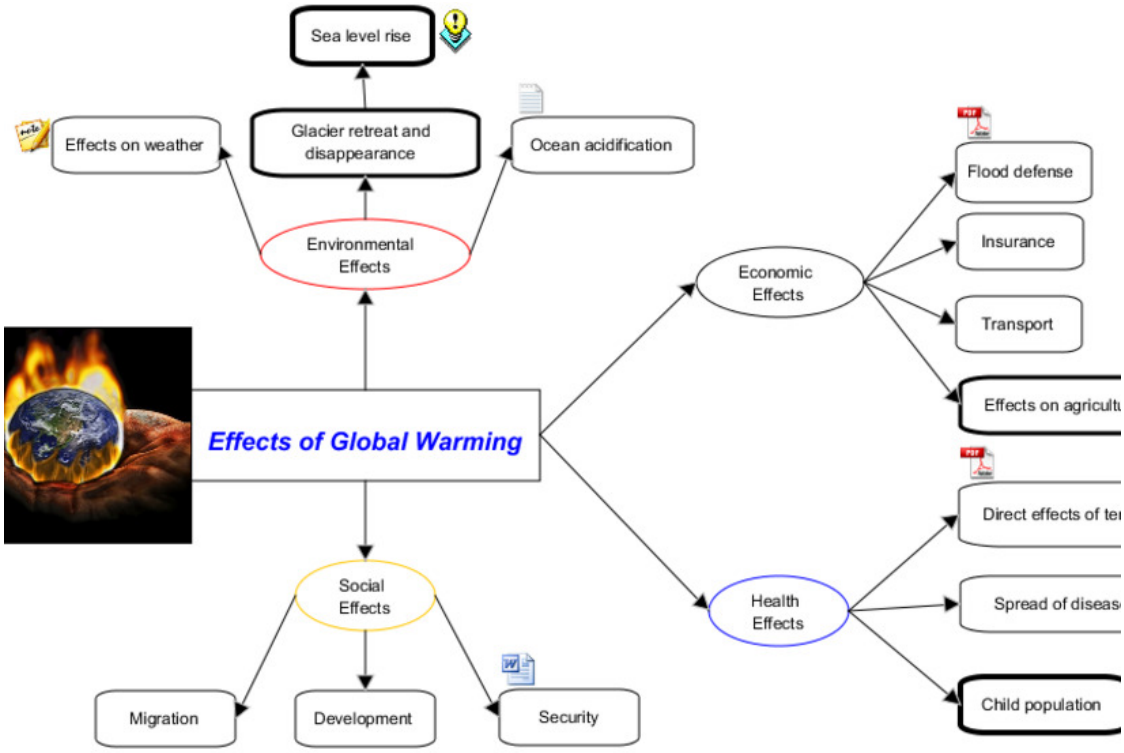
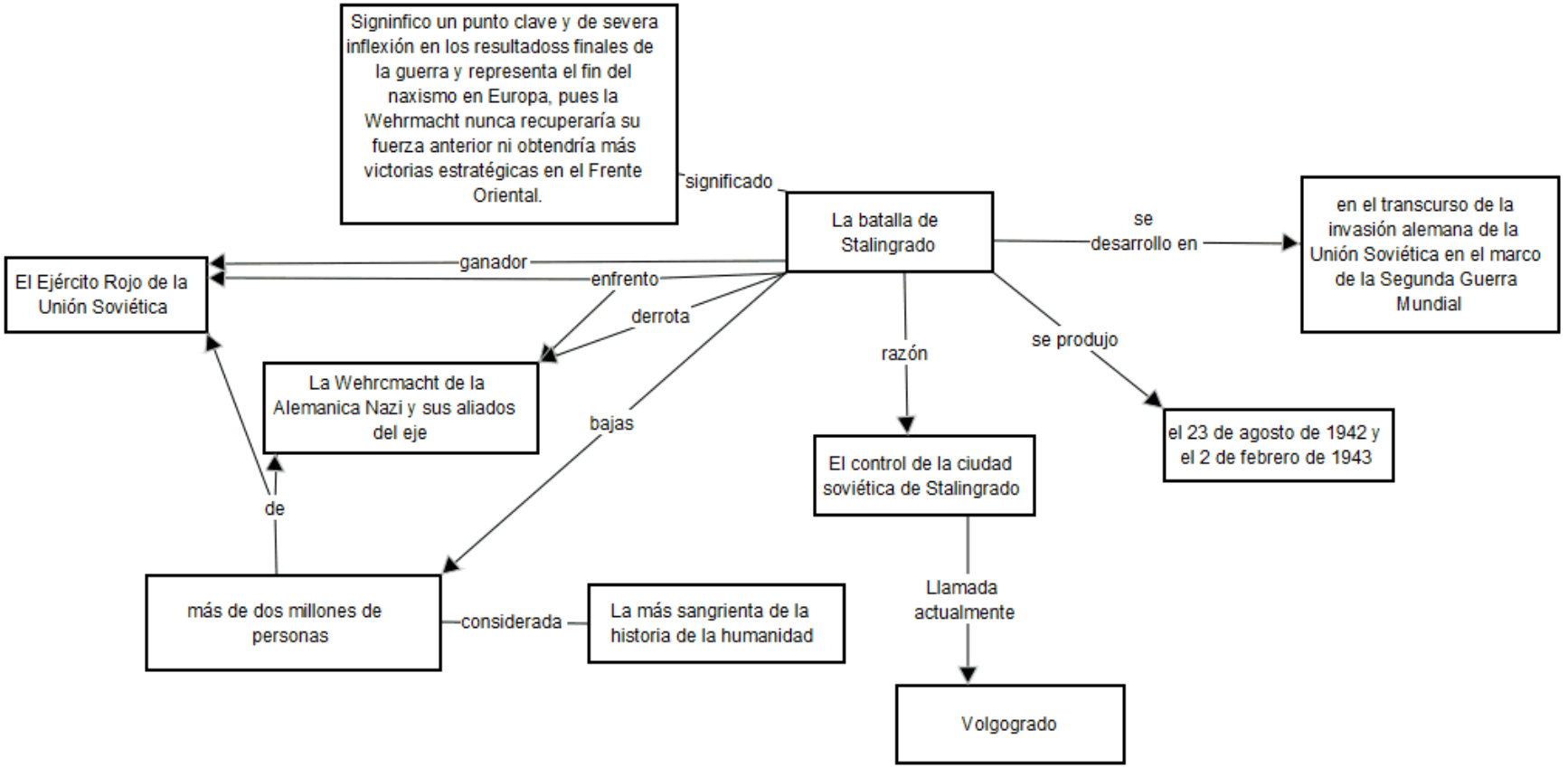


Figura 8-2: Mapa conceptual generado por CM

Su uso en este proyecto ha sido exclusivamente de crear mapas conceptuales a partir de párrafos, con la finalidad de obtener una visión de cómo descomponer un párrafo en un mapa conceptual.

A continuación un ejemplo formado por un párrafo con cuatro sentencias.

La batalla de Stalingrado fue un enfrentamiento bélico entre el Ejército Rojo de la Unión Soviética y la Wehrmacht de la Alemania nazi y sus aliados del Eje por el control de la ciudad soviética de Stalingrado, actual Volgograd, entre el 23 de agosto de 1942 y el 2 de febrero de 1943. La batalla se desarrolló en el transcurso de la invasión alemana de la Unión Soviética en el marco de la Segunda Guerra Mundial. Con bajas estimadas en más de dos millones de personas entre soldados de ambos bandos y civiles soviéticos, la batalla de Stalingrado es considerada la más sangrienta de la historia de la humanidad. La grave derrota de la Alemania nazi y sus aliados en esta ciudad significó un punto clave y de severa inflexión en los resultados finales de la guerra y representa el principio del fin del nazismo en Europa, pues la Wehrmacht



Durante la creación de este proyecto se han conseguido generar preguntas a nivel de sentencias, el siguiente paso sería el de ser capaces de generar preguntas a partir de párrafos, esto quiero decir que las preguntas tendrían una mayor profundidad con respecto al texto. Su complejidad también es mayor ya que entra en juego la semántica, no obstante el analizador basado en dependencias es capaz de analizar párrafos enteros y correlacionarlos entre sí, aparte hay herramientas PNL que son capaces de analizar la semántica, esto nos abre un abanico de posibilidades que pueden aplicarse al mapa conceptual.

8.3. **Posibles mejoras**

Incluso habiendo conseguido unos resultados satisfactorios, ha habido algunas ideas que por falta de tiempo no se llegaron a implementar y que seguramente habrían dado un buen resultado.

- Mejoras para el generador de preguntas a partir de un analizador morfosintáctico.

  - La búsqueda de preguntas parte de la idea principal de que todas las preguntas estén formadas por una estructura que tenga un Sintagma nominal seguido de un Verbo. Este aspecto delimita el número de preguntas que se puedan encontrar dentro de una sentencia. Una posible mejora sería que se realizara la búsqueda de la pregunta sin que esta esté delimitada por ninguna estructura.

  - Por falta de tiempo las preguntas del tipo Dónde no consiguieron realizarse, una de las maneras con la que se pensó en conseguirlas era mediante la herramienta Ixa-pipe-nerc. Esta herramienta es capaz de detectar entidades, y una de estas entidades son las localizaciones.

  - En varias ocasiones el analizador morfosintáctico analizaba las sentencias de manera incorrecta, algo que habría valido la pena saber era el porcentaje de sentencias analizadas correctamente por el analizador. Se podría elaborar un estudio sobre esto y de esta forma darle una mayor credibilidad a los algoritmos creados a partir de estos análisis.

- Mejoras para el generador de preguntas a partir de un analizador basado en dependencias sintácticas.

  - Al igual que en el anterior analizador, la búsqueda de preguntas parte de la idea inicial de que se encuentre un complemente circunstancial dentro del análisis realizado. Si se pudiese obviar este tipo de idea y realizar una búsqueda exhaustiva se conseguiría encontrar una mayor cantidad de preguntas sobre un texto.

  - En general se deberían trabajar más los algoritmos relacionados con este generador, sobre todo las preguntas del tipo cuándo.

Bibliografía

1. **Carbonell., Jaime.** Centro Virtual Cervantes. [En línea] 8 de octubre de 1992. [Citado el: 5 de 10 de 2015.] http://cvc.cervantes.es/obref/congresos/sevilla/tecnologias/ponenc_carbonell.htm.

2. **Hancox, Peter.** School of Computer Science. [En línea] 26 de Enero de 1996. [Citado el: 30 de octubre de 2015.] http://www.cs.bham.ac.uk/~pjh/sem1a5/pt1/pt1_history.html.

3. **Sosa, Eduardo.** El profesional de la información. [En línea] Enero de 1997. [Citado el: 30 de octubre de 2015.] http://www.elprofesionaldelainformacion.com/contenidos/1997/enero/procesamiento_del_lenguaje_natural_revisin_del_estado_actual_bases_tericas_y_aplicaciones_parte_i.html.

4. **IEE.** IEEE Xplore Digital. [En línea] [Citado el: 31 de octubre de 2015.] http://ieeexplore.ieee.org/.

5. **Guitiérrez-Rexach, Bosque y.** Fundamentos de sintaxis formal. *Fundamentos de sintaxis formal.* 2009, págs. 461-463.

6. **Vasile Rus, Brendan Wyse, Paul Piwek, Mihai Lintean, Svetlana Stoyanchev, Cristian Moldovan.** Research Gate. [En línea] 2010.

7. **Bernhard, D., De Viron, L., Moriceau, V., & Tannier, X.** Dialogue & Discourse. [En línea] 2012. http://journals.linguisticsociety.org/elanguage/dad/article/view/2151.html.

8. **Liu, M., Calvo, R. A., & Rus, V.** Dialogue & Discourse. [En línea] 2012. http://dad.uni-bielefeld.de/index.php/dad/article/view/1463.

9. **Aldabe I., Gonzalez-Dios I., Lopez-Gazpio I., Madrazo J., Maritxalar M.** Sociedad Española para el Procesamiento de Lenguaje Natural. [En línea] 2013. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4877.

10. **López Gazpio, Iñigo.** [En línea] 03 de Junio de 2013. https://addi.ehu.es/bitstream/10810/10186/1/Seneko.pdf.

11. **Madrazo Azpiazu, Jon.** addi. [En línea] 25 de septiembre de 2013. https://addi.ehu.es/bitstream/10810/10702/1/Memoria_IonMadrazoAzpiazu.pdf.

12. **Boyer, K.E. and P. Piwek.** Proceedings of the 3rd Workshop on Question Generation. [En línea] 18 de Junio de 2010.

13. **Olney, A. M., Graesser, A. C., & Person, N. K.** Dialogue & Discourse. [En línea] 2012. http://dad.uni-bielefeld.de/index.php/dad/article/view/1480/2832.

14. **Rodrigo Agerri, Josu Bermudez and German Rigau.** Ixa Pipes. [En línea] 2014. http://ixa2.si.ehu.es/ixa-pipes/papers/ixa-pipes-lrec2014.pdf.

15. **Grupo Eagles96.** Eagles. [En línea] http://www.cs.upc.edu/~nlp/tools/parole-sp.html.

16. **Joan Aparicio Mera, Ester Arias Valor, Oriol Borrega Cepa, Patricia fernández, Difda Monterde, Aina Peris Morant, Lourdes Puiggrós Casals, Marta Recasens Potau, Bàrbara Soriano Bautista, Rita Zaragza Jové.** AnCora. [En línea] 2003. http://clic.ub.edu/corpus/es/ancora.

17. **Rus and Graesser.** Question Generation. [En línea] 2009. http://www.questiongeneration.org/.

18. **Carletta, Jean.** Computational Linguistics. [En línea] 1996.

19. **Chang, Allan.** statstodo. [En línea] https://www.statstodo.com/.

20. **GaLan research group.** Galan. [En línea] http://galan.ehu.es/Galan/cmedExperiment.

Apéndice

Apéndice 1

- Código del pre-procesamiento del Corpus.
  - Main

```
public class mainSimp {
	public static void main(String[] args) throws IOException {
		File dir = new File("src/texto/test");
		String[] ficheros = dir.list();
		if (ficheros == null){
			System.out.println("No hay fichero en el
directorio");
		}else{
			for(int i = 0; i < ficheros.length; i ++){
				System.out.println(ficheros[i]);
				Simp simplificar = new Simp(ficheros[i]);
				simplificar.simp2(ficheros[i]);
			}
		}
	}
}
```

- Constructor y simplificador 1: Se tratan los errores de referencias y signos numéricos.

```
public Simp(String file) throws IOException{
            String cadena;
            boolean val = true;
            boolean escribir = true;
            boolean signP = false;
            boolean signC = false;
            boolean cond = true;
            FileWriter fw = new FileWriter("src/texto/test2/"+file);
            FileReader f = new FileReader("src/texto/test/"+file);
            BufferedReader b = new BufferedReader(f);
            int ant = 0;
            while((cadena = b.readLine()) != null){
                  for (int i = 0; i < cadena.length(); i++){
                        System.out.print(cadena.charAt(i));
                        if(cadena.charAt(i) == '(' ){
                              escribir = false;
                        }
                        if(cadena.charAt(i) == ')' ){
                              escribir = true;
                        }
                        if(cadena.charAt(i) == '.' ){
                              signP = true;
                        }else if(cadena.charAt(i) == ',' )        {
                              signC = true;
                        }
                        else if (signP == true || signC == true){
                              if ((int)cadena.charAt(i)>48
                              && (int)cadena.charAt(i)<58
                              || cadena.charAt(i) == ' ' ){
                                    cond = false;
                              }else{
                                    cond = true;
                                    if(signC == true) fw.write(' ');

                                    signC = false;
                                    signP = false;
                              }
                        }
                        if (((int)cadena.charAt(i)>48
                        && (int)cadena.charAt(i)<58
                        && ((int)cadena.charAt(i-1)>96
                        && (int)cadena.charAt(i-1)< 123))
                        || ((int)cadena.charAt(i)>48
                        && (int)cadena.charAt(i)<58
                        && (int)cadena.charAt(i-1)>48
                        && (int)cadena.charAt(i-1)<58
                        && ((int)cadena.charAt(i-2)>96
                        && (int)cadena.charAt(i-2)< 123)) ){
                              val = false;
                        }else{
                              val = true;
                        }

      }
```

```
                        if (escribir == true && cond == true
                        && val==true){
                              if (cadena.charAt(i)== ')'
                              || cadena.charAt(i) == '«'
                              || cadena.charAt(i) == '»'
                              || (int)cadena.charAt(i) == 39
                              || cadena.charAt(i) == '-'){}
                              else
                                    fw.write(cadena.charAt(i));

                        }
                  }
                  fw.write("\r\n");
            }
            fw.close();
            b.close();

      }
```

- Simplificador 2: Se encarga de dejar un espacio después de cada sentencia, ya que la herramientas Ixa pueden detectar a dos sentencias como una si después del punto no hay espacio.

```
public void simp2 (String file) throws IOException{
            String cadena;
            FileReader f = new FileReader("src/texto/test2/"+file);
            FileWriter fw = new FileWriter("src/texto/test3/"+file);
            BufferedReader b = new BufferedReader(f);
            while((cadena = b.readLine()) != null){
                  for (int i = 0; i < cadena.length(); i++){
                        System.out.print(cadena.charAt(i) );

                        if ((int)cadena.charAt(i) >64
                        && (int)cadena.charAt(i) < 123){
                              if(i>1 && cadena.charAt(i-1) == '.'){
                                    fw.write(' ');
                              }
                        }

                        fw.write(cadena.charAt(i));

                  }
                  fw.write("\r\n");

            }
            fw.close();
            b.close();
      }
```

Apéndice 2

- Código del algoritmo Qué del generador de preguntas a partir de un analizador morfosintáctico.

```
public boolean esQue() {
int n;
int m = 0;
//Situarnos en la sentencia a partir de la información
//que tenemos de la palabra(su posición)
for(int i = 0; i < tam; i ++){
      if (listV[i] == true){
            infSN(i);
            buscador.situarEnSentXPos(buscador.
            getInfTipo()[infSel].getPosTexto(),infSel);
            if(buscador.tieneConjuncion() == false){

                  //si  el SN tiene un Determinante
                  //artículo femenino se puede formar la
                  //pregunta qué.
                  if(buscador.tieneDeterminante()){
                        if(buscador.tieneDAF()){
                              indice[0][e] = infSel;
                              e++;
                              indReal[infSel] = i;
                        }else{
                              break;

                        }
                  }
            }
            //Si ha pasado de aquí significa que no tiene
            //ningun determinante al lado y que el grupo
            //nominal está solo

            //buscar el grupo nominal y situarnos en él
            for (m = 0; m < buscador.getAuxSA().getNueva()
            .length; m++){
                  if (buscador.getAuxSA().getNueva()[m].
                  getTipo().compareTo("GRUP.NOM") ==0){

                        buscador.setAuxSA(buscador.
                        getAuxSA().getNueva()[m]) ;
                        break;
                  }
            }

            //puede que ese grupo nominal este formado por
            //otros subgrupos nominales, nos interesa
            //llegar al grupo nominal principal (raiz)
            buscador.situarEnHoja("GRUP.NOM");
```

```
                    //buscar en el grupo nominal el nombre, si este
                    //no se encontrara damos por mal formada la
                    //sentencia
                    for(n = 0; n < buscador.getAuxSA().getNueva().
                    length; n++){
                          if (buscador.getAuxSA().getNueva()[n].
                          getTipo().toCharArray()[0] == 'N'){
                                break;

                          }
                    }
                    if (n >= buscador.getAuxSA().getNueva().
                    length){
                          break;
                    }
                    // si esl nombre es comun, se podrá formar una
                    //pregunta del tipo qué
                    if (buscador.getAuxSA().getNueva()[n].
                    getTipo().
                    toCharArray()[1] == 'C'){
                          indice[0][e] = infSel;
                          e++;
                          indReal[infSel] = i;
                    }

              }
        }
        if (e > 0) {
              e = 0;
              return true;
        }

        return false;
}
```

Apéndice 3

- Código del algoritmo Quién/Quiénes del generador de preguntas a partir de un analizador morfosintáctico.

```
public boolean esQuien() {
int n;
int m = 0;
//Situarnos en la sentencia a partir de la información
//que tenemos de la palabra(su posición)
for(int i = 0; i < tam; i ++){
      boolean entro = false;
      if (listV[i] == true){
            //mirar si tiene un SN a la Izq
            infSN(i);
            buscador.situarEnSentXPos(buscador.
            getInfTipo()[infSel].getPosTexto(),infSel);

            //mirar si tiene conjunción
            if(buscador.tieneConjuncion()){
                  buscador.plural[infSel] = true;
                  indice[1][e] = infSel;
                  e++;
                  indReal[infSel] = i;

            }
            System.out.println(" VALLOOOOOR " + infSel);
            if(buscador.plural[infSel] == false){
                  //mirar si tiene Determinante artículo
                  //masculino o Determinante posesivo
                  //masculino
                  if(buscador.tieneDAMoDPM(infSel)){
                        indice[1][e] = infSel;
                        indReal[infSel] = i;
                        e++;
                        entro = true;
                  }
                  if(buscador.plural[infSel] == false &&
                  entro == false){
                        //Si ha pasado de aquí significa
                        //que no tiene ningún determinante
                        //al lado y que el grupo nominal
                        //está solo.
                        //buscar el grupo nominal y
                        //situarnos en él
                        for (m = 0; m < buscador.
                        getAuxSA().getNueva().length; m++){
                              if (buscador.getAuxSA().
                              getNueva()[m].getTipo().
                              compareTo("GRUP.NOM") ==0){
                              buscador.setAuxSA(buscador.
                              getAuxSA().getNueva()[m]) ;
                                    break;
                              }
                        }
```

```
                                    //puede que ese grupo nominal este
                                    //formado por otros sub grupos
                                    //nominales, nos interesa llegar
                                    //al grupo nominal principal
                                    //(raiz)
                                    buscador.situarEnHoja("GRUP.NOM");
                                    //buscar en el grupo nominal el
                                    //nombre, si este no se encontrara
                                    //damos por mal formada la
                                    //sentencia
                                    for(n = 0; n < buscador.getAuxSA().
                                    getNueva().length; n++){
                                          if (buscador.getAuxSA().
                                          getNueva()[n].getTipo().
                                          toCharArray()[0] == 'N' ){
                                                break;

                                          }
                                    }
                                    if (n >= buscador.getAuxSA().
                                    getNueva().length){
                                          break;
                                    }
                                    //si el nombre es propio, se podrá
                                    //formar una pregunta del tipo
                                    //quién
                                    if (buscador.getAuxSA().
                                    getNueva()[n]. getTipo().
                                    toCharArray()[1] == 'P'){
                                          indice[1][e] = infSel;
                                          e++;
                                          indReal[infSel] = i;
                                    }
                              }
                        }
                  }
            }
            if (e > 0 ){
                  e = 0;
                  return true;
            }

            return false;
      }
```

Apéndice 4

- Código del algoritmo Cuándo del generador de preguntas a partir de un analizador morfosintáctico.

```
	public boolean esCuando() {
	//Situarnos en la sentencia a partir de la información
	//que tenemos de la palabra(su posición)
	for(int i = 0; i < tam; i ++){
		if (listV[i] == true){
			//buscamos en la sentenca analizada el tipo W
			//(fecha)
			if (buscador.tieneTipoW(i)){
				indice[2][e] = i;
				e++;
			}
		}
	}
	if (e > 0){
		e = 0;
		return true;
	}

	return false;
}
```

Apéndice 5

- Código del algoritmo Dónde del generador de preguntas a partir de un analizador basado en dependencias sintácticas.

```
	private boolean esDonde() {
	//buscar si tiene un complemente circustancial
	int i;
	int k = 0;
	int cont = 0;
	boolean tiene = false;
	for ( i = 0; i < listaDep.size(); i++){
		if(listaDep.get(i).getRfunc().compareTo("cc") == 0){
			tieneCC = true;
			cont ++;
			numSuj[0][k] = cont;
		}
		if(tieneCC == true){
			//tiene sujeto
			if(tieneSujetoVinculado(listaDep.get(i),i)){
				//tiene como primer elemento una
				//preposición
				if( listaDep.get(i).getTo().getPos().
				compareTo("P") == 0){
					//comprobar que no es el último
					//elemento
					if(listaDep.size()==i+1){
						break;
					}
					//como siguiente elemento no es una
					//fecha
					if (!esSigFecha(listaDep.
					get(i+1))){
						k++;
						tiene = true;

					}else{
						numSuj[0][k] = -1;
					}
				}
			}else{
				numSuj[0][k] = -1;
			}
		}
		tieneCC = false;
	}
	return tiene;
}
```

Apéndice 6

- Código del algoritmo Cuándo del generador de preguntas a partir de un analizador basado en dependencias sintácticas.

```
	private boolean esCuando() {
	//buscar si tiene un complemente circustancial
	int i;
	int k = 0;
	boolean tiene = false;
	int cont = 0;
	for ( i = 0; i < listaDep.size(); i++){
		if(listaDep.get(i).getRfunc().compareTo("cc") == 0){
			tieneCC = true;
			cont ++;
			numSuj[1][k] = cont;
		}
			if(listaDep.size()==i+1){
				break;
			}
			//como siguiente elemento está clasificado
			//como fecha.
			if(tieneCC == true){
				if(esSigFecha(listaDep.get(i+1))){
					k++;
					tiene = true;
				}else{
					numSuj[1][k] = -1;
				}
			}else{
				numSuj[1][k] = -1;
			}
		tieneCC = false;
	}
	return tiene;
}
```