"Una palabra mal colocada estropea el más bello pensamiento"
Voltaire

A mi familia

Agradecimientos

Es el momento de mostrar mi agradecimiento a todas las personas que han estado ahí y, de una manera u otra, me han ayudado a lo largo del desarrollo de este proyecto, especialmente en los malos momentos.

En primer lugar, a las dos personas, aparte de mi, más involucradas en este proyecto: mi director Manuel Vilares y mi codirectora Mila, que han ayudado a corregir mis errores y me han guiado por el buen camino en los peores momentos. A Manuel Vilares quiero agradecerle también la gran oportunidad que me ha brindado y por la confianza depositada en mi.

Al resto de los compañeros del grupo Cole que en algún momento del desarrollo del proyecto han estado aquí, ayudándome en múltiples ocasiones a superar los problemas con mayor facilidad: Nieves, Juan, Erica, Sara, Vanesa, Víctor, Fran, Mario y Fernando.

A mi familia, especialmente a mis padres José y María del Rosario y a mi hermana Ilda, que han aguantado estoicamente el mal humor con el que llegaba a casa los días que las cosas no me salían bien y por apoyarme como siempre lo han hecho.

Por último, y no por ello menos importante quiero agredecer a todas esas personas que siempre están ahí, dándole sentido a la vida, ayudándome en los malos momentos y estando presentes en la mayoría de los buenos: Natalia, Pablo, Carlos, David, Rafa, Javi, Luis y muchos otros.

A todos vosotros, gracias.

Índice general

I Memoria **13**

1. Introducción **15**
1.1. Identificación del proyecto 15
1.2. Organización de la documentación 15
1.2.1. Memoria 15
1.2.2. Manual técnico 16
1.2.3. Manual de usuario 16
1.3. Origen del proyecto 17
1.4. Objetivos del proyecto 18
1.5. Descripción de la solución empleada 19
1.5.1. Descripción del formato propuesto 19
1.5.2. Formato obtenido 20
1.5.3. Almacenamiento 21
1.5.4. Descripción del sistema 24
1.6. Funcionamiento del sistema 25
1.6.1. Lanzamiento del programa 25
1.6.2. Análisis sintáctico de un texto o frase 26
1.6.3. Gestión de la configuración 26
1.6.4. Gestión de analizadores sintácticos 26

2. Desarrollo del proyecto **29**
2.1. Metodología utilizada 29
2.1.1. El Lenguaje Unificado de Modelado 29
2.1.2. Ciclo de vida 31
2.2. Tecnologías y Herramientas empleadas 33
2.2.1. Tecnologías empleadas 33
2.2.2. Herramientas utilizadas 37

3. Planificación y presupuesto **41**
3.1. Planificación del proyecto 41
3.1.1. Planificación inicial 41
3.1.2. Planificación real 42
3.2. Presupuesto del proyecto 43
3.2.1. Recursos físicos 43
3.2.2. Recursos humanos 44
3.2.3. Coste final 44

4. Conclusiones y posibles ampliaciones **45**
4.1. Dificultades encontradas . . . 45
4.2. Conclusiones . . . 46
4.3. Posibles ampliaciones . . . 46

II Manual técnico **49**

5. Análisis **51**
5.1. Requisitos funcionales . . . 51
5.2. Diagrama de clases . . . 52
5.3. Diccionario de clases . . . 52
5.3.1. Clase configuración . . . 52
5.3.2. Analizador . . . 53
5.3.3. Resultado . . . 54
5.3.4. ResultadoFC . . . 55
5.3.5. ResultadoSinFormato . . . 55
5.3.6. ToResultado . . . 56
5.3.7. ToResultadoImp . . . 56
5.3.8. Item . . . 57
5.3.9. Frase . . . 57
5.3.10. Texto . . . 58
5.4. Casos de uso . . . 58
5.4.1. Caso de uso: Gestión Configuración . . . 59
5.4.2. Caso de uso: Análisis sintáctico . . . 61
5.4.3. Caso de uso: Gestión Analizadores . . . 64

6. Diseño **73**
6.1. Especificación de las clases . . . 73
6.1.1. Paquete Análisis . . . 74
6.1.2. Paquete Resultados . . . 77
6.1.3. Paquete Recursos . . . 79
6.1.4. Paquete Control . . . 80
6.2. Diagramas de secuencia del diseño . . . 83
6.2.1. Caso de uso: Gestión Configuración . . . 83
6.2.2. Caso de uso: Análisis sintáctico . . . 85
6.2.3. Caso de uso: Gestión Analizadores . . . 87
6.3. Patrones de diseño . . . 92
6.3.1. Patrón Singleton . . . 92
6.3.2. Patrón Estrategia . . . 92
6.3.3. Patrón Compuesto . . . 93

7. Implementación y Pruebas **95**
7.1. Implementación . . . 95
7.1.1. Diagrama de componentes . . . 95
7.1.2. Interpretación de los archivos de configuración . . . 96
7.2. Pruebas . . . 99

7.2.1. Pruebas de unidad 99
7.2.2. Pruebas de integración 102
7.2.3. Pruebas de validación 102

III Manual de usuario 103

8. Instalación de la aplicación 105
8.1. Recursos necesarios 105
8.1.1. Recursos hardware 105
8.1.2. Recursos software 105
8.2. Instalación de los programas necesarios 106
8.2.1. Instalación de la máquina virtual Java 106
8.2.2. Instalación del analizador sintáctico Erial 106
8.2.3. Instalación del analizador sintáctico Alpage 106
8.2.4. Instalación de la aplicación 107

9. Uso de la aplicación 109
9.1. Lanzamiento de la aplicación 109
9.2. Análisis sintáctico 111
9.2.1. Lanzar el análisis sintáctico 111
9.2.2. Interpretar los resultados del análisis sintáctico 114
9.3. Gestión de la configuración 116
9.4. Gestión de analizadores sintácticos 117
9.4.1. Crear un nuevo analizador 117
9.4.2. Modificar un analizador 121
9.4.3. Borrar un analizador 122
9.4.4. Editar algoritmos de un analizador 122

IV Apéndices 125

10. Apéndice A: Erial 127
10.1. Introducción 127
10.2. Funcionamiento 127
10.3. Preprocesador 128
10.4. Etiquetador 130
10.5. Analizador sintáctico superficial 131

11. Apéndice B: Cadena de procesamiento lingüístico ALPAGE 133
11.1. Introducción 133
11.2. Funcionamiento 134
11.2.1. Lefff, lexicón sintáctico para el francés 134
11.2.2. sxpipe, una cadena de pre-procesado 134
11.2.3. El lexer easy.pl 135
11.2.4. Dyalog 135
11.2.5. Meta-gramática francesa frmg 136

11.2.6. De la mega-gramática a los parsers . . . 136
11.2.7. Los parsers tig_parser y tag_parser . . . 136
11.2.8. El conversor forest . . . 136
11.2.9. parsed, un servidor de parsers . . . 136
11.3. Instalación . . . 137
11.3.1. Descargar el instalador Alpage . . . 137
11.3.2. ¿Qué se necesita? . . . 137
11.3.3. Opciones . . . 139
11.4. Tutorial . . . 139
11.4.1. Configurar el entorno . . . 139
11.4.2. Uso de los analizadores sintácticos . . . 140
11.4.3. Resolución de problemas . . . 141

Índice de figuras

1.1. Ejemplo de salida del analizador ERIAL 20
1.2. Ejemplo de salida del analizador AVALON 21
1.3. Arquitectura de la aplicación 25

2.1. Ciclo de vida RUP 32

3.1. Planificación inicial 42
3.2. Planificación final 1 42
3.3. Planificación final 2 43

5.1. Diagrama de clases general 52
5.2. Clase Configuración 53
5.3. Clase Analizador 54
5.4. Clase Resultado 55
5.5. Clase DocumentoFC 55
5.6. Clase DocumentoFC 55
5.7. Clase ToResultado 56
5.8. Clase ToResultado 57
5.9. Clase Item 57
5.10. Clase Frase 58
5.11. Clase Texto 58
5.12. Diagrama de casos de uso general 59
5.13. Diagrama de casos de uso Gestión configuración 59
5.14. Diagrama de secuencia: Consultar configuración 60
5.15. Diagrama de secuencia: Editar configuración 61
5.16. Diagrama de casos de uso Análisis sintáctico 62
5.17. Diagrama de secuencia: Analizar frase 62
5.18. Diagrama de secuencia: Analizar texto 64
5.19. Diagrama de secuencia: Transformar a formato común 65
5.20. Diagrama de casos de uso: Gestión analizadores 66
5.21. Diagrama de secuencia: Alta analizador 67
5.22. Diagrama de secuencia: Baja analizador 68
5.23. Diagrama de secuencia: Modificación analizador 69
5.24. Diagrama de secuencia: Consulta analizador 70
5.25. Diagrama de secuencia: Seleccionar analizadores 71
5.26. Diagrama de secuencia: Editar algoritmos 72

6.1. Diagrama de clases general . . . 73
6.2. Paquete Análisis . . . 74
6.3. Paquete Resultados . . . 77
6.4. Paquete recursos . . . 79
6.5. Paquete Control . . . 80
6.6. Diagrama de secuencia de diseño: Consultar configuración . . . 83
6.7. Diagrama de secuencia de diseño: Editar configuración . . . 84
6.8. Diagrama de secuencia de diseño: Analizar frase . . . 85
6.9. Diagrama de secuencia de diseño: analizar texto . . . 86
6.10. Diagrama de secuencia de diseño: Alta analizador . . . 87
6.11. Diagrama de secuencia de diseño: Baja analizador . . . 88
6.12. Diagrama de secuencia de diseño: Consulta analizador . . . 89
6.13. Diagrama de secuencia de diseño: Modificación analizador . . . 90
6.14. Diagrama de secuencia de diseño: Editar algoritmos . . . 91
6.15. Patrón singleton . . . 92
6.16. Patrón estrategia . . . 93
6.17. Patrón compuesto . . . 93

7.1. Diagrama de componentes . . . 96

9.1. Ventana Principal de la aplicación . . . 109
9.2. Vista de análisis sintáctico . . . 111
9.3. Formulario de análisis sintáctico . . . 112
9.4. Diálogo de selección de archivo de texto para el análisis . . . 112
9.5. Mensaje de error . . . 113
9.6. Lista seleccionable de analizadores . . . 113
9.7. Casillas de selección de opciones de análisis . . . 113
9.8. Vista de resultados del análisis . . . 114
9.9. Vista de resultado sin convertir al formato común . . . 115
9.10. Edición de la Configuración . . . 116
9.11. Ventana de gestión de analizadores . . . 117
9.12. Crear analizador: paso 1 . . . 118
9.13. Crear analizador: paso 2 . . . 118
9.14. Crear analizador: paso 3 . . . 119
9.15. Crear analizador: paso 4 . . . 120
9.16. Crear analizador: paso 5 . . . 121
9.17. Modificar analizador . . . 121
9.18. Borrar analizador . . . 122
9.19. Editor java . . . 123

10.1. Funcionamiento de Erial . . . 129
10.2. Estructura del Pre-procesador . . . 129

11.1. Cadena de Procesamiento Lingüístico Alpage . . . 134
11.2. Cadena de pre-procesado sxpipe . . . 135
11.3. Servidor de parsers . . . 137
11.4. Vista gráfica de las dependencias . . . 140

Parte I

Memoria

Capítulo 1

Introducción

1.1. Identificación del proyecto

Código del PFC: **ENI - 237**

Título del PFC: **Formato de intercambio para análisis sintáctico**

Alumno: **Adrián Blanco González**

Director del proyecto: **Manuel Vilares Ferro**

Co-directora del proyecto: **Milagros Fernández Gavilanes**

Departamento: **Informática**.

Área: **Ciencias de la Computación e Inteligencia Artificial**.

Titulación: **Ingeniería Informática**.

1.2. Organización de la documentación

La documentación, como parte esencial en el desarrollo de una aplicación informática, representa el contenido descriptivo que plasma el funcionamiento del sistema, la tecnología empleada y el por qué de su uso. Además sirve para detallar de forma inequívoca todo el trabajo realizado, el método seguido y es la base de las tareas futuras y sus posibles ampliaciones.

Para su mejor comprensión, la documentación de este proyecto se dividió en tres bloques claramente diferenciados como son: la memoria, el manual técnico y el manual de usuario, con los contenidos que se detallan a continuación.

1.2.1. Memoria

La memoria consta de los siguientes capítulos:

- Introducción: Donde se describe el proyecto y se reflejan los objetivos y la descripción técnica del mismo. Además se incluye una breve descripción de la solución adoptada para su desarrollo.

- Desarrollo del proyecto: En este capítulo se muestran los datos estimados y reales sobre la planificación temporal y costes finales.

- Tecnologías y herramientas empleadas: En este capítulo se describen de forma breve las diversas tecnologías y herramientas *software* empleadas en el desarrollo de este proyecto.

- Planificación y presupuesto: Aquí se presenta la planificación inicial realizada para el proyecto y se compara con el desarrollo final del mismo. También se detalla el presupuesto necesario para llevarlo a cabo, teniendo en cuenta tanto los recursos físicos como los humanos.

1.2.2. Manual técnico

El manual técnico está compuesto por los siguientes capítulos:

1. Análisis: Contiene la especificación de los requerimientos de la aplicación desarrollada y la descripción de todos los casos de uso y clases, así como los diagramas de secuencia y de colaboración obtenidos a través de los escenarios.

2. Diseño de la aplicación: Refinamiento de los diagramas y especificaciones realizadas en el análisis, obteniendo un mayor nivel de especificación del sistema.

3. Implementación y pruebas: Este capítulo contiene la descripción de la solución adoptada para este proyecto, junto con ejemplos de código fuente y las pruebas realizadas para verificar la integración, la validez y eficiencia del sistema.

4. Problemas, conclusiones y posibles ampliaciones: Contiene una relación de los problemas encontrados durante el desarrollo del proyecto, las ventajas que aporta el sistema implementado y las posibles ampliaciones que se podrían acometer en un futuro.

5. Apéndices técnicos: En él se documentan las tecnologías o aspectos técnicos más relevantes que se usaron durante el desarrollo del proyecto.

1.2.3. Manual de usuario

Este manual ofrece una descripción del funcionamiento y del uso de la aplicación desarrollada. Incluye toda la información necesaria para llevar a cabo la instalación del sistema, así como una serie de instrucciones para que su uso sea el adecuado. Está compuesto por los siguientes apartados:

1. Requisitos *hardware* y *software*: descripción de los requisitos mínimos necesarios para el correcto funcionamiento de la aplicación.

2. Instalación y ejecución: descripción de los pasos a seguir para lograr su puesta en marcha y del proceso de instalación.

3. Como usar la aplicación: descripción de un modo detallado del uso de la aplicación por parte de un usuario.

1.3. Origen del proyecto

Un sistema de *búsqueda de respuestas* (BR) es aquel capaz de proporcionar una respuesta específica a una pregunta de un usuario. Estos sistemas procesan preguntas arbitrarias y proporcionan respuestas específicas en lenguaje natural tras consultar colecciones de documentos, tales como por ejemplo la *World Wide Web*.

A simple vista, esta actividad no parece muy distinta a la que ya realizan buscadores como *Google* o *Yahoo*, pero hay una diferencia notable: los buscadores mencionados se limitan a devolver una colección de páginas *web* que contienen la información buscada por el usuario, pero, si éste quiere datos concretos, debe recorrer las páginas devueltas para encontrarlos. En cambio, los sistemás de BR devuelven una respuesta precisa en lenguaje natural. Hay que destacar además que en estos sistemas lo primordial es obtener la respuesta, independientemente de qué base de datos o colección es extraída.

Un sistema de búsqueda de respuestas se compone, de manera general, de las siguientes fases:

- Análisis de la pregunta: Para poder responder a la pregunta es necesario conocer ciertas características como el tipo de pregunta o el tipo esperado de respuesta. Por ejemplo, se puede querer conocer un hecho concreto (preguntas factuales) u obtener la definición de un concepto (preguntas de definición).

- Recuperación de información: Una vez conocida la pregunta se debe obtener un subconjunto de la colección de documentos inicial formado por los documentos en los cuales es probable encontrar la respuesta.

- Selección de pasajes: A partir de los documentos obtenidos en el apartado anterior, se debe extraer los pasajes (frases o conjuntos de frases) que puedan contener la respuesta a la pregunta formulada

- Extracción de respuestas: Tomando como punto de partida las frases obtenidas en el apartado anterior, se seleccionan las palabras que forman la respuesta.

El hecho de proporcionar como salida respuestas exactas y no colecciones de documentos, provoca que sea necesario el uso de métodos de *procesamiento del lenguaje natural* (PLN) más complejos que otros sistemas de recuperación de información.

Existe gran variedad de herramientas y métodos para realizar las actividades relacionadas con el PLN, como por ejemplo el *análisis sintáctico*. Por ello, sería interesante disponer de una herramienta que permita obtener los datos en un formato común, independientemente del analizador sintáctico empleado, así como comparar los resultados obtenidos por las diferentes herramientas disponibles de análisis sintáctico.

1.4. Objetivos del proyecto

La finalidad principal de este proyecto es el diseño e implementación de un formato de intercambio común para el desarrollo de una plataforma de BR sobre colecciones documentales en un lenguaje natural. El sistema permitirá parametrizar el acceso a la función de análisis, posibilitando la definición de un entorno integrado de prueba para este tipo de aplicaciones. De un forma más concreta, podemos dividir el proyecto en los siguientes objetivos:

- El primer objetivo es establecer el nuevo formato. Para ello se realizará un estudio de distintas aplicaciones de procesamiento de lenguajes naturales (*lematización*, *etiquetación*, etc.) con el fin de establecerlos aspectos clave en estas aplicaciones, así como un estudio de los diferentes protocolos de interfaz en los analizadores sintácticos disponibles en este tipo de aplicaciones. Para más información sobre la obtención del formato, consultar el apartado 1.5.1.

- Una vez establecido el formato, el siguiente paso será el desarrollo de la aplicación en sí. Ésta, en primer lugar, deber permitir analizar sintácticamente una frase o un texto utilizando un analizador determinado. Los resultados podrán ser mostrados según el formato propuesto en este proyecto o según el formato propio del analizador sintáctico utilizado en el análisis. Al lanzar el análisis se podrán especificar diversos parámetros, como se puede ver a continuación:

  - El analizador sintáctico a utilizar. El usuario podrá, en todo momento, añadir analizadores sintácticos, como se explicará más adelante.
  - El formato de salida. Se podrá mostrar el resultado del análisis siguiendo la salida estándar del analizador utilizado o siguiendo el formato
  - Si se va a medir el tiempo o no. La aplicación permitirá evualuar el rendimiento de cada analizador sintáctico en función del tiempo necesario para llevar a cabo un análisis.

- También se debe permitir al usuario llevar a cabo una completa gestión de los analizadores sintácticos. Concretamente, el usuario podrá realizar las siguientes acciones:

  - Añadir nuevos analizadores sintácticos especificando los comandos y opciones necesarias para lanzarlo, así como características de su formato propio para poder convertirlo al formato de la aplicación.
  - Consultar analizadores sintácticos previamente añadidos.
  - Modificar las características básicas de los analizadores sintácticos existentes en el sistema.
  - Borrar un analizador sintáctico.
  - Modificar los algoritmos utilizados para convertir la salida en el formato propio del analizador al formato común propuesto en este proyecto.
  - Seleccionar los analizadores que se emplearán en los siguientes análisis sintácticos.

1.5. Descripción de la solución empleada

En este apartado se describe de la forma más clara posible la solución adoptada para resolver el problema planteado en este proyecto. Por un lado se detalla el formato escogido para la representación de los resultados de los análisis sintácticos, justificando su elección, y, por otro lado, se describe la arquitectura y funcionamiento de la aplicación.

1.5.1. Descripción del formato propuesto

El objetivo principal de este proyecto es proponer un formato común para mostrar los resultados del análisis sintáctico de frases y textos. La confección de este formato fue un proceso costoso, durante el cual se llevaron a cabo las siguientes actividades:

- Se realizó un estudio exhaustivo de los distintos analizadores existentes con el fin de extraer similitudes e identificar los formatos de salida más completos para tomarlos como ejemplos a seguir.
- Se analizaron diversas herramientas de *Recuperación de Información* (RI) con el fin de identificar los aspectos importantes de los resultados de los análisis sintácticos, para poder incorporarlos al formato propuesto.
- Se estudiaron otras iniciativas similares de obtener una representación normalizada para recursos lingüísticos. Dentro de estas iniciativas destaca el MAF (*Marco para Anotación Morfosintáctica*)[12].
- Se estudiaron diferentes tecnologías para el almacenamiento de los resutados. como por ejemplo *bases de datos* (BD), archivos de texto plano, ficheros XML, etc.

Herramientas estudiadas

Las herramientas revisadas para obtener una visión del formato que se quiere desarrollar han sido las siguientes:

- Erial: Erial (*Extracción y Recuperación de Información mediante Análisis Lingüístico*) es un analizador sintáctico desarrollado por un grupo de investigación de la Universidad de A Coruña (Grupo Cole), las Universidades de Santiago de Compostela y de Vigo y el centro Ramón Piñeiro para la investigación en Humanidades. Se puede encontrar más información sobre Erial en la sección 10.
- Alpage: La cadena de procesamiento desarrollada por el equipo Alpage[1] con el fin de obtener una completa cadena de procesamiento lingüístico para el francés. Su estudio proporcionó una visión sobre el formato a desarrollar, que tomó varios elementos del formato de salida estándar de esta cadena.Para más información sobre esta cadena, consultar la sección 11.

[1] http://alpage.inria.fr/

- MrTagoo[11]: Etiquetador y lematizador de alto rendimento basado en *modelos ocultos de markov* (HMM) de segundo orden que, además, incorpora ciertas capacidades como una estructura muy eficiente para almacenamiento y búsqueda basada en autómtas finitos, gestión de palabras desconocidas y la posibilidad de integrar diccionarios externos. Fue desarrollado por investigadores del grupo Cole.

- TreeTagger: *TreeTagger* es una herramienta para la anotación de texto con información de categoría gramatical y lema que ha sido desarrollado dentro del proyecto TC en el *Institute for Computational Linguistics of the University of Stuttgart*. Ha sido empleado exitosamentepara etiquetar textos en alemán, inglés, francés, italiano, español, griego, y francés antiguo, y es fácilmente adaptable a otros lenguajes si se dispone de un lexicon y corpus marcado manualmente.

1.5.2. Formato obtenido

Del estudio de los distintos analizadores sintácticos se han extraído las similitudes entre las salidas de los mismos. Se ha concluído que la información más significativa que se debe considerar a la hora de crear el nuevo formato es la siguiente:

- Frase: Es fundamental que la salida contenga la frase en su forma original, ya que facilita enormemente la comprensión de la misma.

- Palabras: Para cada palabra que compone la frase se deben almacenar datos importantes como pueden ser el lema o la categoría gramatical.

- Dependencias: Por último, es una constante en las salidas de los analizadores observados información sobre las relaciones de dependencia entre las distintas palabras, indicando el tipo de dependencia con una etiqueta.

Veamos, por ejemplo, un gráfico que representa la salida que produce el analizador sintáctico Erial al analizar la frase *"Ella come una manzana"*: Para un analizador

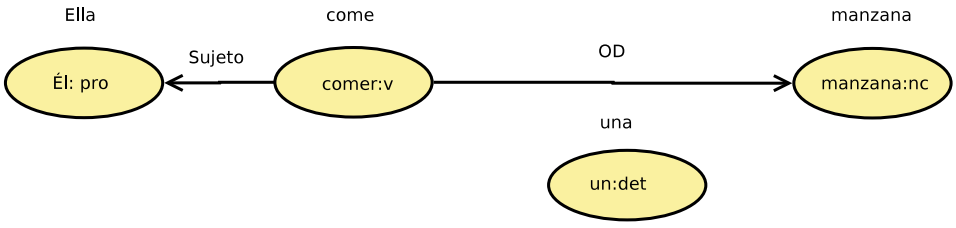

Figura 1.1: Ejemplo de salida del analizador ERIAL

sintáctico como Erial, que sólo ofrece un análisis sintáctico, este formato es apropiado. Sin embargo, otros analizadores, como Alpage o Avalon muestran en sus salidas todos los análisis posibles, siendo necesario etiquetar una misma palabras de varias formas distintas. Para que el formato sea válido para todos los casos es necesario modificar la estructura propuesta. Para ello se introducirá el concepto de clúster, que contendrá todos las opciones de análisis contempladas para una palabra

de la frase. Cada una de estas opciones estará representada por un nodo del grafo. Veamos como quedaría estructurado este formato:

- Frase: Al igual que la propuesta anterior, la salida contiene información de la frase en su forma original.
- Clusters: Cada cluster almacena, para cada palabra, la propia palabra tal y como aparece en la frase y la posición que ocupa en la misma. Un cluster está compuesto, a su vez, por uno o más nodos.
- Nodos: Un nodo representa una opción de análisis para una palabra de la frase.La información que se almacena para cada nodo es la siguiente: categoría gramatical, lema y cluster al que pertenece.
- Dependencias: En este caso, las dependencias serán entre nodos. Para cada dependencia se almacenan los nodos involucrados y el tipo de dependencia.

A continuación se muestra la salida que produciría el analizador AVALON con la frase del ejemplo anterior:

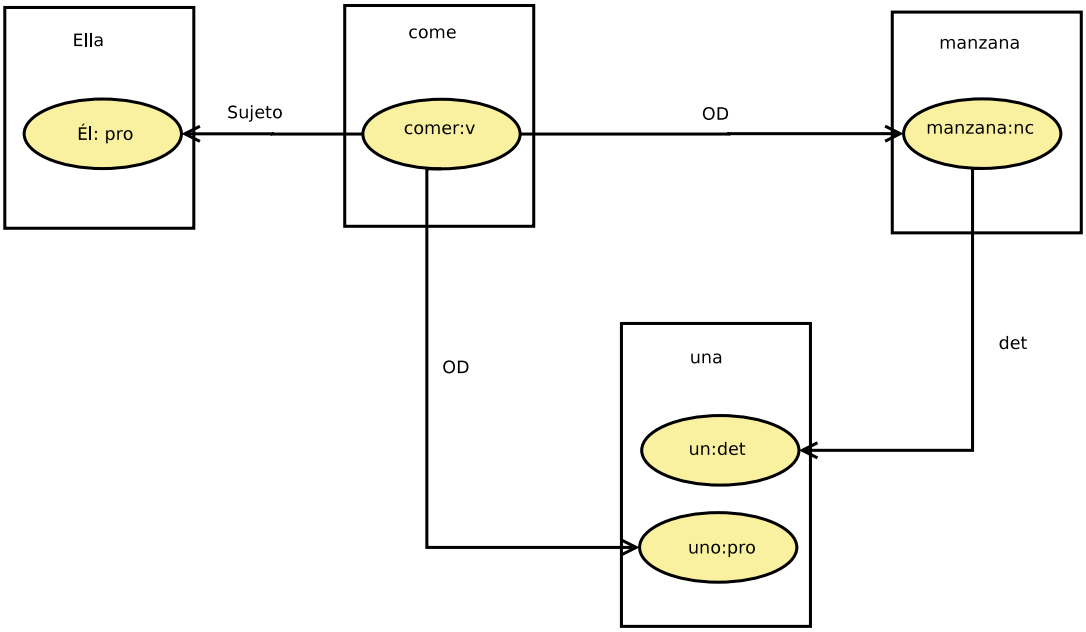

Figura 1.2: Ejemplo de salida del analizador AVALON

1.5.3. Almacenamiento

Se ha optado por la tecnología XML para el almacenamiento de los resultados debido a las siguientes razones:

- Se ha convertido en un estándar.

- La mayoría de los lenguajes de programación, en especial *Java* (usado en este proyecto), proporcionan librerías para manejar archivos de este tipo.
- Es fácilmente entendible para los usuarios.

Un DTD (*Definición de Tipo de Documento*) es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD.A continuación se puede ver el DTD al que deben ajustarse los resultados expresados en el formato propuesto.

```
<!ELEMENT frase (dependencia | nodo | cluster)*>
<!ATTLIST frase
valor CDATA #IMPLIED
>

<!ELEMENT cluster EMPTY>
<!ATTLIST cluster
lex CDATA #IMPLIED
token CDATA #IMPLIED
fin CDATA #IMPLIED
inicio CDATA #IMPLIED
id CDATA #IMPLIED
>

<!ELEMENT nodo EMPTY>
<!ATTLIST nodo
id CDATA #IMPLIED
lema CDATA #IMPLIED
cluster CDATA #IMPLIED
cat CDATA #IMPLIED
>

<!ELEMENT dependencia EMPTY>
<!ATTLIST dependencia
id CDATA #IMPLIED
origen CDATA #IMPLIED
destino CDATA #IMPLIED
tipo CDATA #IMPLIED
etiqueta CDATA #IMPLIED
peso CDATA #IMPLIED
>
```

A continuación vamos a describir brevemente los componentes del anterior DTD. En primer lugar, está el elemento *frase*, que contiene un atributo llamado *valor* que contiene la frase completa que se ha analizado. El elemento frase contiene, a su vez, varios elementos *cluster*, *nodo* y *dependencia*, que se describirán a continuación. El elemento *cluster* representa el concepto de cluster explicado en el apartado anterior y está compuesto por los siguientes atributos:

- *lex*: Contiene el *lexema* de la palabra a la que se refiere el *cluster*.
- *token*: Contiene la palabra tal y como aparece en la frase.
- *inicio*, *fin*: Delimitan la posición de la palabra en la frase.
- *id*: Identificador del *cluster*.

El elemento *nodo* representa a cada una de las opciones de análisis de la palabra. Tiene los siguientes atributos:

- *id*: Identificador del nodo.
- *lema*: Contiene la palabra en su forma canónica, tal y como aparece en el diccionario.
- *cluster*: Identificador del *cluster* al que pertenece este nodo.
- *cat*: Categoría gramatical que se le asigna a la palabra.

El elemento *dependencia* representa la relación entre los distintos nodos. Contiene los siguientes atributos:

- *id*: Identificador de la dependencia.
- *origen*, *destino*: Nodos relacionados mediante la *dependencia*.
- *tipo*: Tipo de *dependencia*.
- *etiqueta*: Etiqueta que se le asigna a la *dependencia*.
- *peso*: Probabilidad de que la *dependencia* se corresponda con el análisis real de la frase.

Por último veamos un ejemplo de como quedaría la representación en XML del resultado de un análisis sintáctico expresado según las pautas marcadas por el formato propuesto. A continuación se muestra en formato XML el resultado previamente presentado en formato gráfico en la figura 1.1:

```
<?xml version="1.0"encoding="UTF-8"?>

<frase valor="null">
<cluster id ="c_1_2" inicio="1" fin="2"token="come" lex="come" />
<cluster id = "c_2_3" inicio="2" fin="3" token="una" lex="una" />
<cluster id ="c_0_1" inicio="0" fin="1" token="Juan" lex="Juan" />
<cluster id ="c_3_4" inicio="3" fin="4 t"oken="manzana" lex="manzana" />

<nodo cat="v" cluster="c_1_2" lema="comer" id="n002" />
<nodo cat="det" cluster="c_2_3" lema="un" id="n003" />
<nodo cat="pro" cluster="c_0_1" lema="Juan" id="n001" />
<nodo cat="nc" cluster="c_3_4" lema="manzana" id="n004" />

<dependencia origen="n002" destino="n001" etiqueta="sujeto" tipo=" "
id="e001" peso="1" />
<dependencia origen="n002" destino="n004" etiqueta="OD" tipo=" "
id="e002" peso="1" />

</frase>
```

1.5.4. Descripción del sistema

En este apartado se expone la solución adoptada, así como las características de la aplicación desarrollada.

Como se ha mencionado anteriormente, el objetivo de este proyecto es desarrollar una aplicación que permita utilizar distintos analizadores, transformando sus salidas al formato descrito en el apartado 1.5.1. Aunque los analizadores sintácticos con los que se ha trabajado y que están incluidos en la aplicación son Erial, Alpage y Freeling, el sistema se ha llevado a cabo considerando en todo momento la posibilidad de añadir nuevos analizadores.

A continuación se detallan las funcionalidades específicas de cada uno de los módulos de la aplicación desarrollada, junto con una breve descripción del flujo de información entre las distintas partes. Se trata de mostrar un primer acercamiento al sistema, posteriormente en los capítulos dedicados al análisis, diseño e implementación, la descripción será más detallada.
A continuación se describirán brevemente los módulos que componen la arquitectura de la aplicación:

- Interfaz: La interfaz permite al usuario comunicarse con la aplicación. Este módulo maneja toda la lógica relacionada con el conjunto de vistas presentadas al usuario para la realización de las tareas que pretenda lleva a cabo. Según las opciones elegidas, el sistema mostrará una vista u otra, ofreciendo la posibilidad al usuario de introducir los datos que se le solicite para ejecutar una tarea.

- Control: Este módulo es el responsable de controlar la configuración del sistema y comunicarse con los distintos componentes, sirviendo de puente entre la

interfaz y el módulo de análisis, enviando peticiones al mismo cuando se quiere realizar un análisis sintáctico y gestionando los distintos analizadores.

- Análisis: El módulo del análisis es el que lleva a cabo los análisis sintácticos. Trabaja con documentos, en principio de texto y genera los resultados en formato XML.

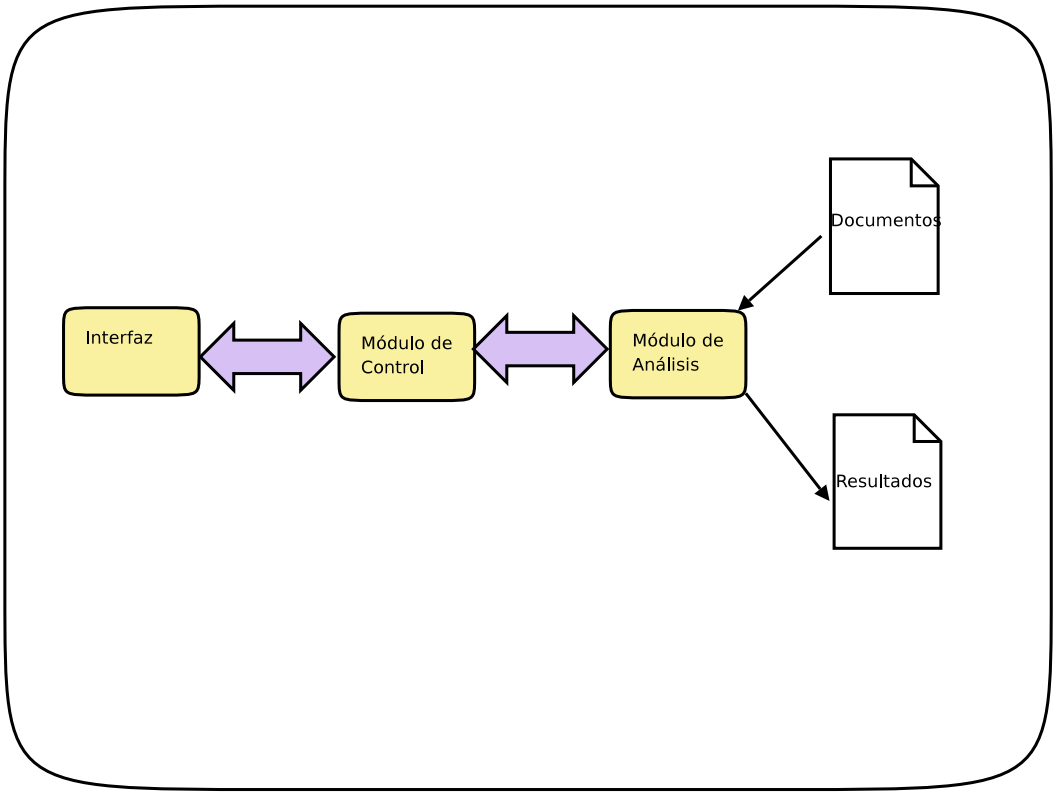

Figura 1.3: Arquitectura de la aplicación

1.6. Funcionamiento del sistema

En este apartado se presenta un ejemplo del funcionamiento del sistema.

1.6.1. Lanzamiento del programa

Lo primero que hace el programa tras ser lanzado es cargar el archivo de configuración. El sistema analiza dicho fichero para encontrar posibles errores o subsanar ciertas inconsistencias, por ejemplo, crear un directorio si este no existe. Una vez hecho esto, el usuario podrá llevar a cabo diversas tareas.

1.6.2. Análisis sintáctico de un texto o frase

El objetivo principal de este proyecto es permitir llevar a cabo análisis sintáctico de frases y textos utilizando diversos analizadores disponibles. Eso se ha de hacer de la siguiente manera:

- Lo primero es seleccionar el analizador o analizadores que se van a emplear. La aplicación permite lanzar varios análisis al mismo tiempo con distintos analizadores, mostrando de forma muy intuitiva los resultados.
- A continuación se deben especificar algunos parámetros importantes para el análisis sintáctico. Se debe establecer qué formato se debe utilizar para mostrar la salida (el propio del analizador sintáctico o el formato común propuesto) y si se debe medir el tiempo empleado por cada analizador para llevar a cabo el análisis o no.
- Lo último que se debe hacer antes de lanzar el análisis es introducir la frase o texto a analizar. Esto se puede hacer de las siguientes maneras:
  - Si lo que se quiere es analizar una frase, lo más recomendable es introducirla directamente por teclado.
  - Si, por el contrario, se quiere analizar un texto compuesto por varias frases, se debe introducir la ruta del fichero en el que se en cuentra dicho texto.

  Una vez echo esto se puede lanzar el análisis.

1.6.3. Gestión de la configuración

Como se ha explicado anteriormente, al lanzar la aplicción se carga un fichero de configuración, que contiene una serie de parámetros fundamentales para el funcionamiento de la misma. Es muy importante, por lo tanto, que este fichero pueda modificarse según las necesidades del usuario. Si bien se puede modificar manualmente (es un fichero XML) la manera más segura es mediante un sistema de formularios con comprobación de errores que proporciona la aplicación. De esta manera nos aseguramos que el fichero contenga siempre valores correctos.

1.6.4. Gestión de analizadores sintácticos

El sistema, como ya se ha mencionado en otros apartados, permitirá realizar una completa gestión de los analizadores sintácticos.

- Si lo que se quiere es añadir un nuevo analizador sintáctico, se deben ir introduciendo los datos necesarios. El sistema irá verificando estos datos e informando al usuario de los errores que se produzcan.
- Si se quiere borran un analizador, se deberá seleccionar dicho analizador entre la lista de todos los analizadores existentes.

- Si se quiere consultar información relativa a un analizador, se seleccionará el analizador sintáctico concreto que se quiera consultar y sus datos serán mostrados al usuario.
- Si se quiere modificar los datos de un analizador sintáctico, se debe hacer lo siguiente: Una vez seleccionado éste, el usuario tendrá la oportunidad de modificar sus datos.
- Por último, si se quiere modificar los algoritmos del analizador, se mostrará un sencillo editor Java que permitirá al usuario modificar dichos algoritmos.

Capítulo 2

Desarrollo del proyecto

2.1. Metodología utilizada

En este apartado se detallan diversos aspectos relacionados con el desarrollo del proyecto, como son la metodología empleada para el análisis y el diseño así como el modelo de proceso seguido.

2.1.1. El Lenguaje Unificado de Modelado

El análisis y diseño del sistema se desarrolló utilizando el *Lenguaje Unificado de Modelado* (UML: Unified Markup Language). Este lenguaje surgió a partir de diversos métodos de análisis y diseño orientados a objetos a finales de la década de los 80 y principios de los 90. Unifica los métodos de Rumbaugh, Jacobson y Booch [1]. Hoy en día, UML está consolidado como el lenguaje estándar en el análisis y diseño de sistemas. Mediante UML es posible establecer la serie de requerimientos y estructuras necesarias para plasmar un sistema de *software* previo al proceso intensivo de desarrollo de código. A pesar de ser UML un lenguaje y de que muchas herramientas hoy en día permitan generar código a partir de especificaciones UML, posee más características visuales que programáticas.

Cuanto más complejo es el sistema que se desea crear, más beneficios presenta el uso de UML. Esto se debe a dos puntos básicos:

1. Mediante una visión global resulta más fácil detectar las dependencias y dificultades implícitas del sistema.

2. Los cambios en una etapa inicial resultan más fáciles de llevar a cabo que en una etapa final del sistema.

A continuación algunas características de UML:

- El sistema de *software* es diseñado y documentado antes de que sea codificado.

- Los lógicos errores en la etapa de diseño podrán ser detectados con anterioridad. El *software* se comportará de la forma esperada y surgirán menos imprevistos.

- El diseño total del sistema dicta el modo en que se desarrollará el *software*. Las decisiones finales se harán antes de que se encuentre código mal escrito. Con esto se ahorra tiempo en el desarrollo.

- Cualquier modificación en el sistema será más fácil de llevar a cabo sobre la documentación UML que sobre el propio código.

En el proceso de desarrollo del proyecto se han utilizado los siguientes diagramas:

- Diagrama de casos de uso: modela el funcionamiento esperado de la aplicación. Más detalladamente, modela los distintos requisitos funcionales que se esperan de una aplicación y cómo se relacionan entre ellos.

- Diagrama de clases: es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos.

- Diagrama de secuencia: es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo.

- Diagrama de paquetes: muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Modelado del sistema

A continuación se describen las especificaciones y diagramas realizados en las distintas etapas de desarrollo del sistema.

Análisis

Esta etapa está descrita en el capítulo 5 del manual técnico. El primer paso para realizar un buen análisis consiste en realizar una buena captura y especificación de requisitos. Esta fase, además de ser de vital importancia para el devenir del proyecto, es muy compleja, ya que los usuarios muchas veces no saben claramente lo que quiren. Una vez identificados y priorizados, los requisitos han de ser especificados formalmente, mediante los correspondientes diagramas de casos de uso y de secuencia. Finalmente, se realizará el diagrama de clases inicial del sistema.

Diseño

En el capítulo 6 del manual técnico se detalla el diseño del sistema. En una primera etapa se desarrolla un diagrama de clases correspondiente a cada paquete. A diferencia del diagrama de clases del análisis, éstos aportan un mayor nivel de detalle al modelado del sistema. A continuación se prosigue con los diagramas de secuencia correspondientes , y que aportan una profundidad mayor que los realizados durante el análisis, siendo más cercanos al sistema real implementado. Para completar esta fase, se aplican patrones de diseño que permiten optimizar y facilitar el desarrollo del sistema.

Implementación
En el capítulo de implementación del manual técnico se describirán varios aspectos y tareas llevadas a cabo para la completa y correcta puesta en marcha del sistema. Se aclararán varios aspectos relacionados con el fichero de configuración. Se detallarán las DTD's que dan formato a los distintos ficheros de almacenamiento XML utilizados.

2.1.2. Ciclo de vida

Todo proyecto de software se compone de una serie de actividades. Estas actividades pueden agruparse en fases porque globalmente contribuyen a la obtención de un producto intermedio, necesario para continuar hacia el producto final y para facilitar la gestión del proyecto. A este conjunto de fases se le denomina *ciclo de vida* o *modelo de proceso*. Sus objetivos son los siguientes:

- Definición de las actividades que se deberán llevar a cabo durante el proyecto.
- Proporcionar unos puntos de control y administrativos, de forma que se pueda evaluar el sistema.

Sin embargo, la forma de agrupar las actividades, los objetivos de cada fase, los tipos de productos intermedios que se generan, etc. pueden ser muy diferentes dependiendo del tipo de producto o proceso a generar y de las tecnologías empleadas.

En el caso concreto de este proyecto se ha obtado por el modelo de proceso de desarrollo conocido con el nombre como *proceso unificado de rational* (RUP).

El RUP es uno de los modelos más generales de los que se utilizan actualmente. Es un proceso de ingeniería planteado por Kruchten [2], cuyo objetivo es producir *software* de alta calidad. Trata de cumplir con los requerimientos de los usuarios dentro de una planificación y presupuestos establecidos. RUP está dirigido por casos de uso, centrado en la arquitectura, iterativa (mini-proyectos) e incremental (versiones).

Toma en cuenta las mejores prácticas en el modelo de desarrollo de *software*, en particular las siguientes:

- Adaptar el proceso: El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.
- Balancear prioridades: Los requisitos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos. Debido a este balanceo se podrán corregir desacuerdos que surjan en el futuro.
- Demostrar valor iterativamente: Los proyectos se entregan, aunque sea internamente, en *etapas iteradas*. En cada etapa se evalúa la estabilidad y calidad del producto y se va refinando la dirección del proyecto.

- Elevar el nivel de abstracción: Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o marcos de referencia (*frameworks*) por nombrar algunos. Esto evita que los ingenieros de *software* vayan directamente de los requisitos a la codificación de *software* a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la mejor manera los requerimientos y sin comenzar desde un principio pensando en la reutilización del código. Un alto nivel de abstracción también permite discusiones sobre diversos niveles y soluciones arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

- Enfocarse en la calidad: No sólo la funcionalidad es importante. También lo son el rendimiento y la confiabilidad del sistema. El RUP ayuda a planificar, diseñar, evaluar y ejecutar pruebas que verifiquen estas cualidades durante todas las fases del proceso. Asegurar la calidad del producto final debe ser parte del proceso de desarrollo.

Ciclos y fases

El RUP divide el proceso de desarrollo en ciclos. Se obtiene un producto al final de cada ciclo. Éstos se dividen en cuatro fases:

Figura 2.1: Ciclo de vida RUP

- Inicio: En esta fase se debe establecer una base para poder comparar los gastos reales con los gastos planificados. Si el proyecto no supera este hito, conocido como *"hito del objetivo del ciclo de vida"*, o bien se cancela, o bien se repite después de ser rediseñado para obtener unos mejores resultados.

- Elaboración: En esta fase el proyecto empieza a tomar forma. Se realiza el *análisis del dominio del problema* y la arquitectura del proyecto obtiene su forma básica. Esta fase debe pasar el *"hito de la arquitectura del ciclo de*

vida". Si el proyecto no supera este hito, aún está a tiempo de ser cancelado o rediseñado. Una vez superada esta fase, el proyecto se convierte en una operación de alto riesgo en la que es mucho más dificil responder a los cambios, por lo que es importante asegurarse de que el análisis realizado cumple con los requisitos del cliente.

- Construcción: Esta fase se centra en el desarrollo de componentes y otros rasgos del sistema que se está diseñando. Esta es la fase en la que se realiza la mayor parte de la codificación. En proyectos grandes, se pueden llevar a cabo varias iteraciones con el fin de dividir los casos de uso en trozos más razonables que producen prototipos demostrables. Esta fase produce la primera salida externa del *software*. Su conclusión estará marcada por el *"hito de capacidad operacional"* inicial.

- Transmisión: El objetivo es llegar a obtener una versión del producto para su prueba con los usuarios. Una vez instalado surgirán nuevos elementos que implicaran nuevos ciclos. Esta etapa conlleva la realización de pruebas, entrenar a los usuarios y a los encargados del manteninimento y distribuir el producto.

2.2. Tecnologías y Herramientas empleadas

En el presente capítulo se describirán brevemente las tecnologías y las herramientas empleadas para llevar a cabo este proyecto, destacando características representativas de las mismas, así como los motivos que han provocado su selección frente a otras herramientas y técnicas similares.

2.2.1. Tecnologías empleadas

Java

Java es un lenguaje de programación orientado a objetos desarrollado por *Sun Microsystems* a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de *C* y *C++*, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Esta sencillez, unida a otras características interesantes, como el hecho de ser facilmente portable a diferentes plataformas gracias al concepto de *Máquina Virtual*, han hecho de *Java* uno de los lenguajes de programación más extendidos y populares. A continuación se describen brevemente sus principales rasgos.

- Lenguaje Orientado a Objetos: *Java* fue concebido desde el primer momento como un lenguaje orientado a objetos. La *Programación Orientada a Objetos*(referida en muchas ocasiones mediante sus siglas en inglés, OOP) se basa en el modelado de objetos e interacciones entre ellos para diseñar e implementar programas. Su potencia se basa en el uso de sus cuatro características principales: herencia, polimorfismo, encapsulamiento y abstracción.

- Lenguaje interpretado y compilado: Una característica fundamental de *Java* es el hecho de ser compilado e interpretado. Es compilado, ya que todo programa escrito en *Java* es, mediante compilación, transformado en un código intermedio llamado *bytecodes*. Por otra parte, es interpretado ya que los *bytecodes* se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real.

- Robusto: *Java* fue diseñado para crear *software* altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores: la aritmética de punteros, ya que se ha prescindido por completo de los mismos. La recolección de basura (*garbage collector*) elimina la necesidad de la liberación explícita de memoria.

- Seguro: El hecho de que *Java* sea interpretado también permite hacerlo seguro. Como la ejecución de los programas *Java* está controlada por el intérprete de *Java*, éste puede contener al programa y evitar que provoque efectos no deseados en el sistema.

- Independiente de la arquitectura: *Java* está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos, desde Linux a Windows, pasando por Mac. La independencia de la plataforma es una de las razones por las que *Java* es interesante ya que muchas personas deben poder acceder desde distintos ordenadores. Pero no se queda ahí, *Java* está siendo desarrollado incluso para distintos tipos de dispositivos además del ordenador de sobremesa, como móviles y agendas.

- Portable: Al ser los programas *Java* interpretados en lugar de compilados, resulta más sencillo ejecutarlos en una variedad de entornos.

- Dinámico: El lenguaje *Java* y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas.

XML

XML es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Permite estructurar la información en un documento o, en general, en cualquier fichero que contenga texto, como por ejemplo ficheros de configuración de un programa o una tabla de datos. Ha ganado mucha popularidad en los últimos años por ser un estándar abierto y libre.

A continuación se citan algunas de las principales características de XML, que lo han convertido en un estándar y han hecho que sea utilizado en este proyecto:

- Es extensible, es decir, después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.

- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión del lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan errores y se acelera el desarrollo de aplicaciones.

- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

- La codificación del contenido en XML consigue que la estructura de la información resulte más accesible. Además, la independencia entre el contenido de los datos y la presentación de los mismos hace de éste un formato adecuado para el desarrollo de un sistema como el que se presenta en este proyecto.

- Es un formato ideal para guardar datos de configuración de las aplicaciones.

- Presenta elementos como las DTD's o XMLSchema que permiten verificar de forma sencilla si la estructura del documento se corresponde con la deseada.

- Existen abundantes herramientas y API's para trabajar con información representada en este formato. Estas herramientas facilitan el acceso, tratamiento y modificación de los datos. En concreto el lenguaje de programación *Java*, empleado en la codificación de este proyecto, dispone de varias API's como DOM y SAX, que se verán a continuación.

A lo largo de este proyecto, se ha utilizado XML para los diversos ficheros de configuración y, sobre todo, para el formato que almacena los resultados de los análisis sintácticos.

SAX

SAX son las siglas de *simple* API *for* XML (API simple para XML). Esta librería funciona por eventos y métodos asociados. A medida que el analizador SAX analiza el documento, va encontrando distintos componentes del mismo, es decir, elementos, atributos y valores. Estos componentes son lo que se conoce como *eventos*. La detección de uno de ellos desencadena una acción asociada, conocida con el nombre de *método*. A medida que se recorre el archivo, SAX detecta la presencia de eventos invocando a los métodos que el programador ha desarrollado.

El API consta de una serie de clases con sus correspondientes métodos, que permiten trabajar con un documento XML desde un programa escrito en *Java*, pudiendo acceder a los datos, comprobar si está bien formado, si es válido, etc. Su principal característica es que el fichero es leído secuencialmente de principio a fin, sin tener que cargar todo el documento en memoria.

SAX se ha empleado principalmente en las consultas sobre ficheros XML, debido a su sencillez. En caso de que haya que realizar modificaciones sobre los ficheros se ha optado por usar JDOM.

JDOM

Documento de Modelado de Objetos en Java (JDOM) es una biblioteca para manipulación de datos XML optimizada desde *Java*. A pesar de su similitud con DOM del consorcio *World Wide Web*, es una alternativa como documento para modelado de objetos que no está incluído en DOM. La principal diferencia es que mientras que DOM fue creado para ser un lenguaje neutral e inicialmente usado para la manipulación de páginas HTML con *JavaScript*, JDOM se creó específicamente para usarse con *Java* y, por lo tanto, beneficiarse de las características de *Java*, incluyendo sobrecarga de métodos, colecciones, etc. Para los programadores de *Java*, JDOM es una extensión más natural y correcta. Se asemeja al sistema RMI (invocación remota de métodos) optimizado para *Java*.

En el caso concreto de este proyecto, JDOM se ha utilizado para crear y modificar los resultados de los análisis sintácticos, debido a su sencillez para modificar ficheros en XML y a que no se necesita manejar una cantidad excesiva de datos.

SWING

Swing es una biblioteca gráfica para *Java* que forma parte de las *Java foundation classes* (JFC). Incluye *widgets* para la interfaz gráfica de usuario tales como cajas de texto, botones, árboles, listas desplegables, etc.

Swing existe desde la JDK 1.1. Antes de la existencia de *Swing*, las interfaces gráficas con el usuario se realizaban a través del *Abstract Window Toolkit* (AWT), del que hereda todo el manejo de eventos. Usualmente, para toda componente AWT existe una componente *Swing* que la reemplaza.

Son muchas las ventajas que proporciona el uso de *Swing*. Por ejemplo, la navegación con el teclado es automática, cualquier aplicación *Swing* se puede utilizar sin ratón. todo ello, sin tener que escribir ni una línea de código adicional.

JGraph

JGraph es una librería para la visualización de gráficos en Java. Fue diseñada para ser completamente compatible con Swing (ver 2.2.1), tanto visualmente como en lo relativo a la arquitectura. *JGraph* Funciona con la versión 1.4 de Java y posteriores. Esta librería proporciona una serie de funcionalidades de dibujo de gráficos para aplicaciones. Aunque simple, la API de *JGraph* permite visualizar, interactuar y analizar gráficos. En este proyecto, se ha empleado *JGraph* para mostrar gráficamente los resultados de los análisis sintácticos. Las razones principales por las que se ha elegido *JGraph* frente a otras API's semejantes son las siguientes:

- Flexibilidad: Permite modificar muchos parámetros de visualización y elaborar elementos de visualización propios.

- Documentación: El equipo desarrollador de *JGraph* proporciona una gran cantidad de documentación de ayuda, desde javadoc a manuales y tutoriales, lo que facilita el aprendizaje.

- Compatibilidad con Swing: Como se ha explicado anteriormente, *JGraph* se integra perfectamente con Swing, API empleada para implementar la interfaz gráfica de la aplicación.

LaTeX

LaTeX es un lenguaje de marcado para documentos y un sistema de preparación de documentos, formado por un gran conjunto de macros de TeX, escritas inicialmente por *Leslie Lamport*[3], con la intención de facilitar el uso del lenguaje de composición tipográfica creado por *Donald Knuth*. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. *LaTeX* es *software* libre bajo licencia *Licencia Pública del Proyecto LaTeX* (LPPL). Se ha utilizado *LaTeX* para realizar esta documentación.

2.2.2. Herramientas utilizadas

En esta sección se describen de forma breve las diferentes herramientas utilizadas en la realización de las distintas tareas comprendidas en este proyecto. Para cada una de ellas se explicará con que fin se ha empleado en este proyecto y porqué se ha seleccionado por delante de otras herramientas similares.

NetBeans

NetBeans es una plataforma para el desarrollo de aplicaciones de escritorio, así como un *entorno integrado de desarrollo* (IDE).

La plataforma *NetBeans* permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las API's de *NetBeans* y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma *NetBeans* pueden ser extendidas fácilmente por otros desarrolladores de software.

El IDE *NetBeans* permite escribir, compilar, depurar y ejecutar programas. Está escrito en Java, lenguaje empleado en este proyecto, pero permite ser utilizado con un gran número de lenguajes de programación. Es un producto libre y gratuito, sin restricciones de uso.

Se ha decidido emplear este IDE frente a otros, como por ejemplo *Eclipse* debido a sus muchas posibilidades para generar código automáticamente y depurar programas y a las facilidades que ofrece a la hora de crear interfaces gráficas.

JDK 1.6

Puesto que el desarrollo del sistema se ha llevado a cabo con el lenguaje de programación Java, se hace necesario el uso de un compilador, así como de un

entorno de ejecución.

Java Development Kit se puede definir como un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar aplicaciones en Java, proporcionado por *Sun Microsystems*. Incluye las librerías básicas de Java, el *Java Runtime Environment* (JRE), un compilador y otras funcionalidades definidas por Sun. Proporciona por lo tanto el entorno de ejecución necesario para ejecutar programas hechos en Java.

Visual Paradigm

Visual Paradigm es una herramienta CASE que utiliza UML como lenguaje de modelado (*www.visual-paradigm.com*). La herramienta está diseñada para una amplia gama de usuarios interesados en construir sistemas de software fiables con el uso del paradigma orientado a objetos, incluyendo actividades como ingeniería de software, análisis de sistemas y análisis de negocios. Esta herramienta ha sido utilizada en el desarrollo de este proyecto para la elaboración del análisis y del diseño.

Analizadores sintácticos empleados

En el desarrollo de este proyecto se han empleado distintas herramientas para el análisis sintáctico. Aunque la aplicación permite al usuario añadir nuevos analizadores, se han estudiado e incorporado a la aplicación algunos de ellos. A continuación se muestran los analizadores sintácticos estudiados, así como sus características fundamentales, si bien en la sección **??** se describen más a fondo y se explica cómo deben ser usados.

- Erial: Es uno de los analizadores sintácticos utilizados a lo largo de este proyecto. Fue desarrollado por un grupo de investigación de la Universidad de A Coruña, las Universidades de Santiago de Compostela y de Vigo y el centro Ramón Piñeiro para la investigación en Humanidades. Para más información sobre *Erial* y su uso, consultar la sección 10.

- Alpage: La cadena de procesamiento Alpage fue desarrollada por el equipo *Alpage* con el fin de obtener una completa cadena de procesamiento lingüístico para el francés. Para más información sobre esta cadena y su funcionamiento, consultar la sección 11.

- Freeling: Es una herramienta para análisis ligüístico en varios idiomas, pensada para ser utilizada como librería externa para aplicaciones que necesiten este tipo de procesamiento lingüístico.

- AGFL:Es un sistema para el desarrollo de gramáticas para lenguajes naturales y la generación automática de analizadores sintácticos eficientes a partir de dichas gramáticas.

Kile

Kile es un editor de Tex/LaTeX. Funciona conjuntamente con KDE en varios sistemas operativos. Las principales características por las que se ha seleccionado *Kile* frente a otros editores son las siguientes:

- Autocompletado de comandos LaTeX.
- Coloreado de sintaxis. *Kile* automáticamente marca los comandos LaTeX y resalta los parentesis.
- Permite trabajar con múltiples ficheros simultaneamente.
- Proporciona plantillas y patrones para facilitar la creación de documentos.
- Plegado de código.

Planner

Planner es una herramienta para la gestión de proyectos que permite realizar diagramas de Gantt, asignación de recursos, etc. Forma parte del proyecto *gnome* y se puede instalar desde los repositorios de Ubuntu.

Ubuntu

Ubuntu[1] es una distribución GNU/Linux que ofrece un sistema operativo principalmente enfocado a computadores personales. Es una de las más importantes distribuciones de GNU/Linux. Se basa en Debian y concentra su objetivo en la facilidad y libertad de uso, la fluida instalación y los lanzamientos regulares (cada 6 meses: las versiones .04 en abril y las .10 en octubre). El principal patrocinador es Canonical Ltd., una empresa privada fundada y financiada por el empresario sudafricano *Mark Shuttleworth*. La versión utilizada en este proyecto es la 8.04, lanzada en abril del 2008.

[1] http://www.ubuntu.com/

Capítulo 3

Planificación y presupuesto

3.1. Planificación del proyecto

A continuación se muestran las dos planificaciones realizadas para el desarrollo de este proyecto. La primera de ellas es la planificación prevista y fue realizada antes del inicio del desarrollo del sistema. La segunda mostrada, es la planificación real que se realizó al finalizar el proyecto para poder observar las desviaciones producidas y las causas de las mismas.

3.1.1. Planificación inicial

Las fases planificadas para el desarrollo del proyecto fueron las siguientes:

- Estudios previos: En esta fase se produce un primer acercamiento al sistema, estudiando cuales son los objetivos que se persiguen y las posible maneras de conseguir estos objetivos. Además se incluye en esta fase el estudio de diversas herramientas de PLN, con el objetivo de establecer los aspectos claves en estas aplicaciones, así como los protocolos seguidos por los analizadores para mostrar las salidas en este tipo de aplicaciones, lo que nos permitirá establecer un formato común para el análisis sintáctico, que es el objetivo principal de este proyecto.

- Análisis y diseño: En la etapa de análisis se realiza la especificación de requisitos, los casos de uso, el diagrama de clases inicial y los diagramas de secuencia con el fin de obtener una descripción detallada del sistema, así como capturar todas las especificaciones. Durante la etapa del diseño se profundizan los conceptos del análisis, detallando los diagramas de secuencia, refinando los diagramas de clases y aplicando patrones de diseño.

- Implementación del sistema: En esta fase se incluye la implementación completa del sistema, así como el estudio de las tecnologías necesarias para llevarla a cabo.

- Pruebas. En esta fase se realizan las pruebas finales del sistema, con el objetivo de corregir los posibles errores cometidos durante la implementación.

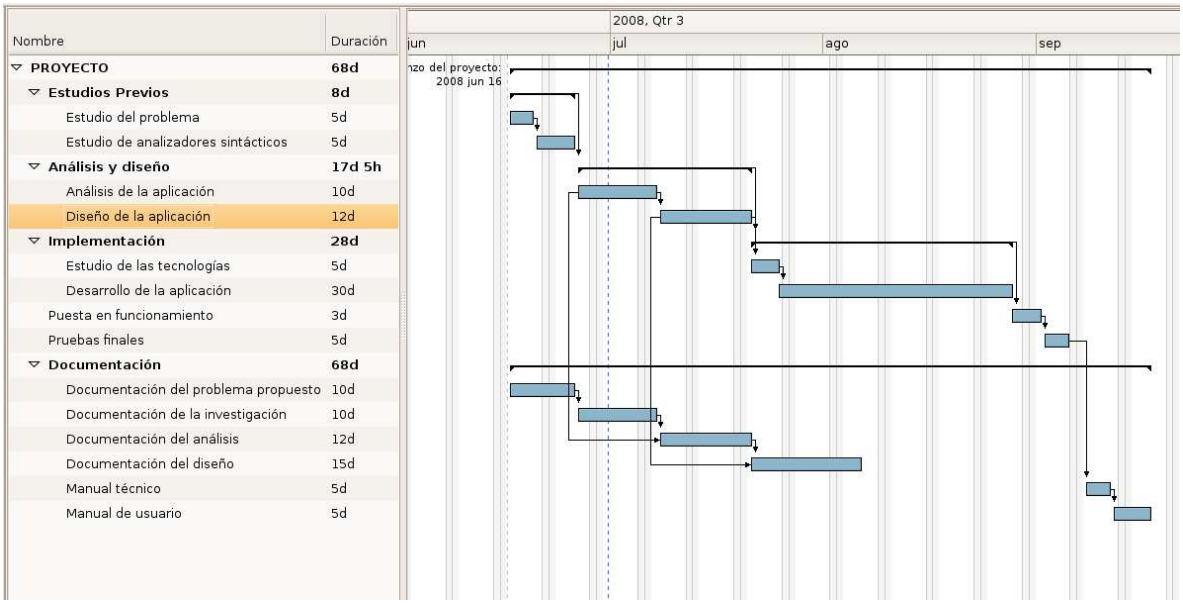

Figura 3.1: Planificación inicial

3.1.2. Planificación real

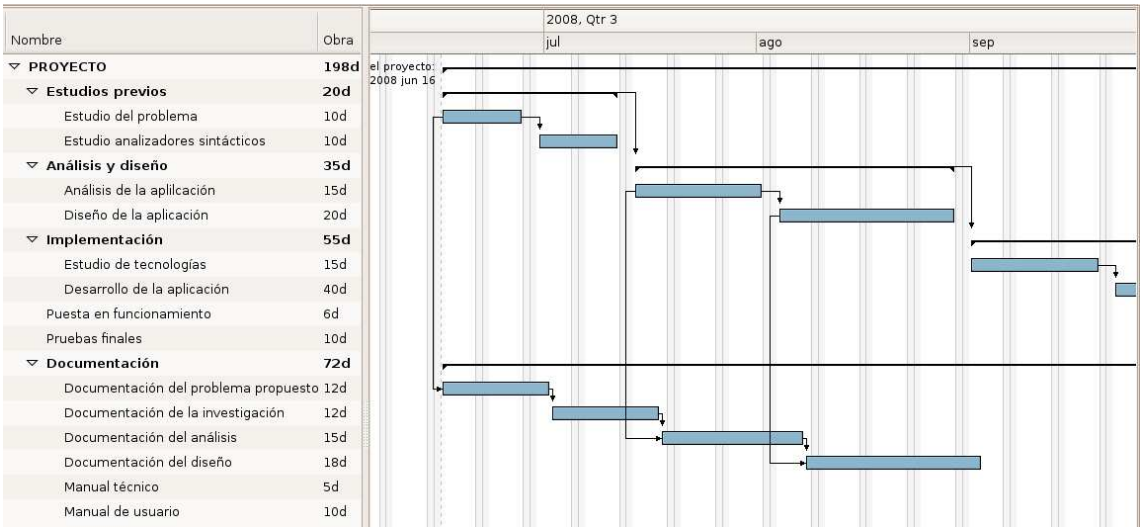

Figura 3.2: Planificación final 1

Se puede observar que, aunque las fases de desarrollo inicialmente previstas siguen siendo las mismas durante el desarrollo del sistema, se produjeron ciertas desviaciones en cuanto a los tiempos estimados al inicio del proyecto. Los motivos de dichas desviaciones son los siguientes:

- Una de las principales causas es la inexperiencia en la realización de planificaciones de este tipo y el excesivo optimismo a la hora de realizar la planificación inicial.
- La fase de estudios previos tuvo una duración mayor que la prevista, ya

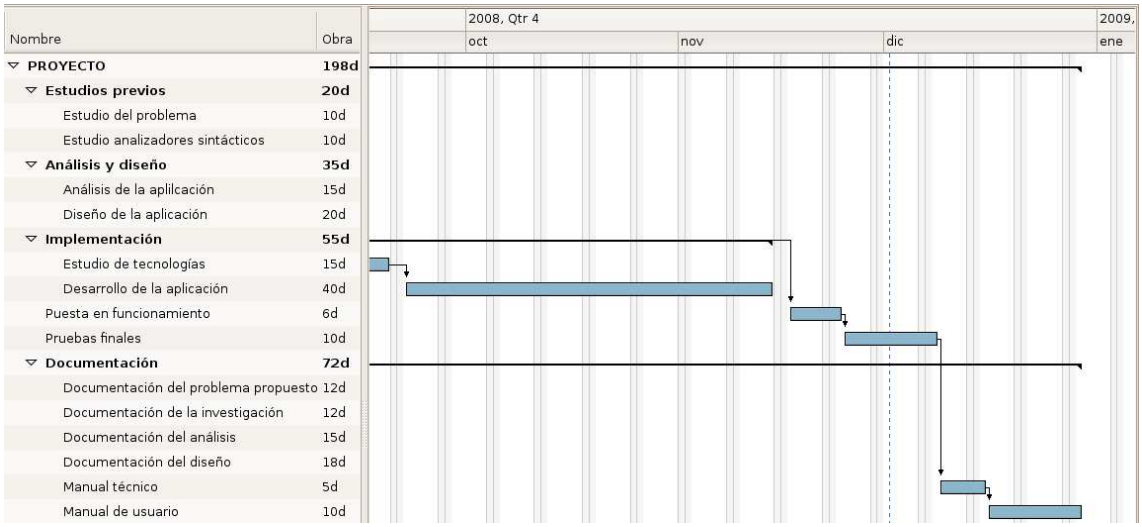

Figura 3.3: Planificación final 2

que muchas de las herramientas consideradas inicialmente no cumplían los requisitos necesarios.

- Las fases de análisis y diseño vieron aumentada su duración respecto a la inicialmente prevista debido a cambios en las especificaciones y a la poca experiencia en desarrollar aplicaciones de este tipo y manejar patrones de diseño.

- La implementación tardó más de lo previsto en llevarse a cabo debido al estudio de diversas librerías. En especial, resultó muy costoso encontrar una librería gráfica que cumpliese con las funcionalidades requeridas.

3.2. Presupuesto del proyecto

En este apartado se detalla el presupuesto necesario para llevar a cabo el proyecto. Para ello se han separado los costes físicos, tanto de *hardware* como de *software*, de los costes referentes a los recursos humanos.

3.2.1. Recursos físicos

Recursos hardware

El *hardware* está amortizado a diez proyectos, como regla general en el desarrollo de aplicaciones.

Hardware	**Coste**
Portatil con Intel Centrino Duo 2 GHz; 2048 Gb RAM; HDD: 200GB	1000 €
TOTAL (coste / 10)	100 €

Recursos software

El hecho de que se haya utilizado exclusivamente *software* libre en el desarrollo del proyecto provoca coste 0 en este apartado:

- Sistema operativo GNU/Linux. Distribución Ubuntu 8.04.
- Editor NetBeans versión 6.0.1 para Ubuntu.
- Herramienta para el modelado: Visual Paradigm community edition para ubuntu, versión libre disponible para alumnos de la *Escuela Superior de Ingeniería Informática* (ESEI) en el campus de Ourense.
- Editor Kile para llevar a cabo la documentación, en LaTeX.
- Gestor de pr oyectos Planner, incluído en Ubuntu. Se ha utilizado para realizar los diagramas de Gantt necesarios.

3.2.2. Recursos humanos

Si bien el desarrollador de este proyecto ha sido una sola persona, dependiendo de la tarea realizada genera costes diferentes.

Recursos Humanos	Duración	Coste planificado
Analista (40 €/hora)	35 días X 6 horas/día	8400 €
Diseñador (35 €/hora)	20 días X 6 horas/día	4200 €
Programador (25 €/hora)	71 días X 6 horas/día	10650 €
Total		23250 €

3.2.3. Coste final

Para hallar el coste final del desarrolo del proyecto se han sumado los costes relativos a los recursos humanos a los costes relativos a los recursos físicos:

Recursos físicos	100 €
Recursos humanos	23250 €
TOTAL	**23350**

Capítulo 4

Conclusiones y posibles ampliaciones

4.1. Dificultades encontradas

A lo largo del desarrollo se han ido encontrando diversas dificultades que se han superado con éxito. Los principales problemas surgieron durante la fase de estudio y durante la fase de implementación. A continuación se detallan cuales han sido los más relevantes y como se han podido solucionar:

- En primer lugar hay que destacar la difícil tarea de investigación llevada a cabo para encontrar, entender y poner en funcionamiento los distintos analizadores sintácticos. En una persona sin experiencia la búsqueda de este tipo de recursos es de suma dificultad, pues se desconocen los sitios donde buscar información sobre los mismos y donde obtenerlos. A esto hay que sumarle el hecho de trabajar con idiomas poco explotados en estos campos, como son el francés y, sobre todo, el español, para los que encontrar aplicaciones de este tipo no es una tarea sencilla.

- Otra dificultad que se encontró al intentar establecer el formato común para la salida de los analizadores sintácticos, ya que hubo que estudiar el comportamiento de varias herramientas para intentar que el formato fuera fácilmente adaptable desde los distintos analizadores. Sin embargo esto no siempre es fácil y algunos analizadores sintácticos han tenido que ser desechados ya que tenían una salida orientada a ser comprensible al ojo humano lo que, en algunas ocasiones, complicaba la extración de información útil y, por lo tanto, la conversión al formato común. Este es el caso, por ejemplo de la salida del analizador *Freeling*

- También se encontraron complicaciones a la hora de implementar la funcionalidad relativa a permitir al usuario añadir nuevos analizadores a la aplicación, ya que ello supone la adición en tiempo de ejecución de nuevas clases que implementen los algoritmos para realizar la llamada al analizador y para convertir su salida al formato común. Para solucionar este problema hubo que estudiar a fondo las posibilidades que ofrece el lenguaje *java* a la

hora de compilar clases en tiempo real e incorporar dichas posibilidades a la aplicación contemplando todos los errores que puediesen surgir de permitir al usuario modificar el código.

4.2. Conclusiones

La principal conclusión a la que se ha llegado una vez finalizado el proyecto es que éste ha servido adentrarse en el mundo del procesamiento del lenguaje natural (PLN). He aprendido muchos conceptos referentes al PLN que no conocía y he tenido que estudiar varias herramientas, aprender a instalarlas y manejarlas, lo que supone una experiencia muy positiva de cara al futuro.

Este proyecto también ha servido para introducirme en el mundo de *LaTeX*, un lenguaje de marcado ideal para documentos técnicos, como es el caso de la documentación de este proyecto. Si bien ya conocía este lenguaje y lo había utilizado muy ocasionalmente, el haber desarrollado la documentación de este proyecto utilizándolo hace que haya adquirido muchos conocimientos al respecto. No obstante, aún me queda mucho que aprender sobre *LaTeX*, lo cual espero hacerlo en el futuro ya que estoy muy satisfecho con el resultado.

Cabe destacar la dificultad de implementar correctamente los patrones de diseño, debido a que, dependiendo del uso que se le vaya a dar existen diversas técnicas de implementación. Por ello, debemos buscar siempre la más idónea, evaluando y realizando pruebas que determinen cual es la que más se aproxima a las necesidades que se requieren.

Uno de los retos más importantes que se han encontrado a la hora de realizar este proyecto, ha sido el gran número de tecnologías y herramientas empleadas: XML, Java, herramientas y tecnologías estándar en el ámbito del procesamiento de lenguaje natural (PLN) y en el de la recuperación de información (RI). Fue necesario dedicar cierto tiempo en el estudio y el manejo de estas tecnologías, sobre todo en lo que respecta al uso de las diversas herramientas como *Erial*, *Alpage*, *Freeling* o *Agfl*.

El hecho de incorporar tantas herramientas provocó una serie de complicaciones durante el desarrollo de la aplicación. A esto hay que sumarle mi inexperiencia en cuanto a desarrollo de aplicaciones que tienen que interactuar con otro *Software* (en este caso, los distintos analizadores), lo que provocó que el encontrar errores fruto de esa interacción fuera, en muchos casos, una tarea ardua.

Por último cabe destacar la importancia de realizar pruebas de todo tipo durante todo el proceso de desarrollo de *software*, ya que si dejamos estas pruebas para el final, la corrección de los errores encontrados se volvería mucho más difícil.

4.3. Posibles ampliaciones

A continuación se plantean una serie de posibles ampliaciones al sistema desarrollado:

- Aunque se han probado e integrado en la aplicación algunos analizadores, no estaría de más incorporar un mayor número de herramientas de análisis o,

en su defecto, facilitar al usuario la incoroporación de nuevos analizadores sintácticos, acercando la aplicación a usuarios que no dominen el lenguaje *Java* (imprescindible a la hora de crear los algoritmos necesarios para realizar el análisis).

- Mejorar la interfaz gráfica del usuario, sobre todo en lo referente a mostrar los resultados del análisis, permitiendo al usuario personalizar la vista gráfica de dichos resultados.

- Otra posible ampliación sería ofrecer la posibilidad de combinar los resultados obtenidos por los distintos analizadores al realizar el análisis sintáctico sobre la misma frase, obteniendo posibles salidas ambigüas con los correspondientes pesos para cada opción de análisis.

Parte II

Manual técnico

Capítulo 5

Análisis

La fase de análisis es la más importante de todo el proceso de desarrollo de software, hasta el punto de que un buen análisis representa un 70 % del resultado final. En ella se capturan los requisitos, tanto funcionales como no funcionales, y se especifican de la manera más clara posible, mediante el uso de diagramas y descripciones textuales.

5.1. Requisitos funcionales

A continuación se muestran los requisitos o funcionalidades que debe cumplir el sistema. Estós son el punto de partida del desarrollo del proyecto. El objetivo final del mismo es obtener un producto que satisfaga dichos requisitos.

- Analizar una frase
- Analizar un texto
- Convertir frase analizada al formato común
- Convertir documento analizado al formato común
- Añadir un nuevo analizador al sistema
- Eliminar un analizador del sistema
- Editar un analizador existente
- Consultar un analizador existente
- Seleccionar analizador/es
- Editar código fuente
- Visualizar el resultado de un análisis como texto
- Visualizar el resultado de un análisis gráficamente
- Editar la configuración del sistema
- Consultar la configuración del sistema

5.2. Diagrama de clases

A continuación se muestra el *diagrama de clases* general del sistema:

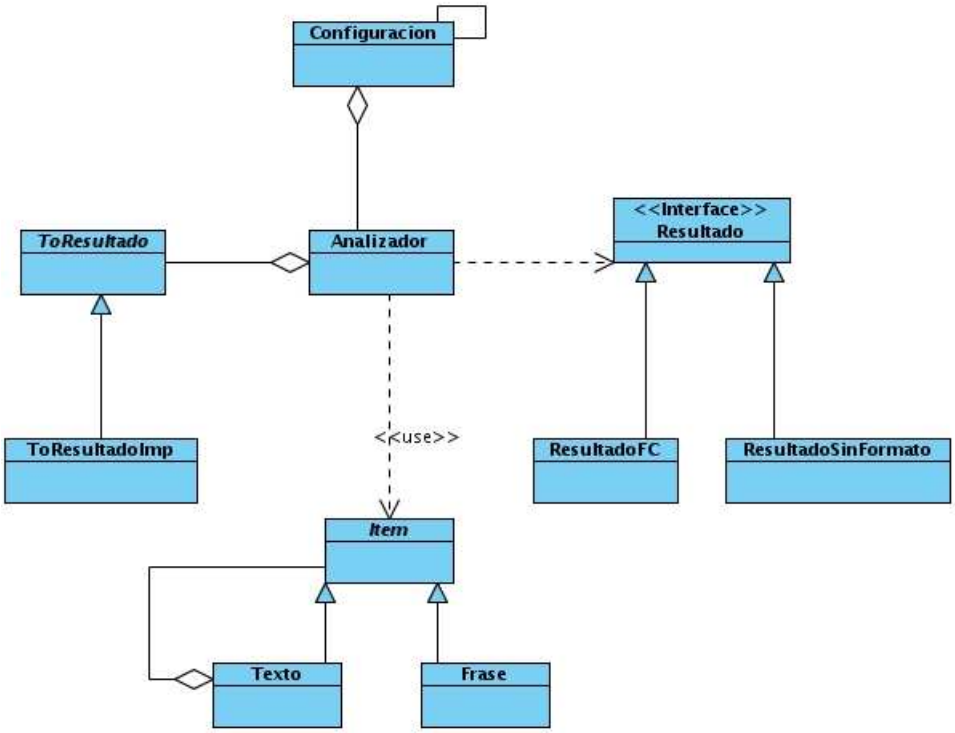

Figura 5.1: Diagrama de clases general

5.3. Diccionario de clases

5.3.1. Clase configuración

Esta clase maneja toda la información necesaria para el funcionamiento de la aplicación, permitiendo configurarla para obtener los resultados deseados.

Atributos

- propiedades: características fundamentales que definen el comportamiento de la aplicación.
- analizadores: todos los analizadores sintácticos presentes en el sistema.
- analizadores_seleccionados:
  analizador o analizadores seleccionados actualmente para realizar el análisis sintáctico.

Figura 5.2: Clase Configuración

Métodos

- get/setPropiedad(): métodos que permiten obtener, modificar las propiedades del sistema.
- load(): método que permite cargar la configuración desde el archivo de configuración.
- store(): método que permite almacenar la configuración actual en el archivo de configuración.
- loadDefault: método que permite cargar la configuración por defecto.
- configure(): método que permite aplicar la configuración actual.
- crear(): añade un nuevo analizador sintáctico al sistema.
- actualizar(): actualiza un analizador sintáctico existente.
- eliminar(): elimina un analizador existente del sistema.
- seleccionar(): selecciona un analizador o conjunto de analizadores para realizar el análisis sintáctico.

5.3.2. Analizador

En esta clase se mantiene la información relativa a los analizadores sintácticos y se gestionan las llamadas a los mismos, así como los resultados obtenidos.

Atributos

- nombre: nombre único del analizador sintáctico que lo identifica y diferencia de los demás analizadores.
- directorio_base: directorio base del analizador.

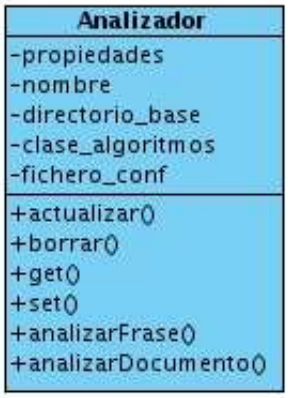

Figura 5.3: Clase Analizador

- fichero_conf: fichero XML de configuración del analizador.
- propiedades: características generales que definen el comportamiento de un analizador
- clase_algoritmos: clase que contiene la implementación de los algoritmos para analizar un ítem y pasarlo a formato común para este analizador en concreto.

Métodos

- actualizar(): actualiza las características del analizador.
- borrar(): elimina un analizador existente del sistema.
- get(): devuelve una propiedad del analizador.
- set(): establece/modifica una propiedad del analizador.
- analizarFrase(): analiza sintácticamente una frase.
- analizarDocumento(): analiza sintácticamente un documento de texto.

5.3.3. Resultado

Esta interfaz mantiene toda la información relativa a un ítem (texto o frase) analizado sintácticamente, independientemente del formato escogido para su almacenamiento.

Métodos

- leer(): lee un documento analizado desde un archivo.
- escribir(): escribe un documento analizado en un archivo.
- mostrar(): muestra el contenido de un documento analizado.

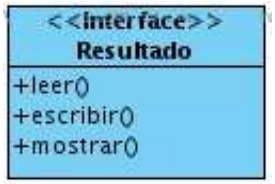

Figura 5.4: Clase Resultado

5.3.4. ResultadoFC

Esta clase mantiene toda la información relativa a un ítem (texto o frase) analizado sintácticamente siguiendo las pautas establecidas por el formato propuesto en este proyecto.

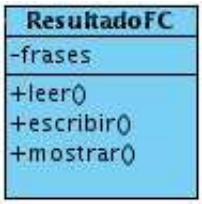

Figura 5.5: Clase DocumentoFC

Atributos

- frase: resultado obtenido por el análisis agrupado en frases.

Métodos

- leer(): lee un documento analizado desde un archivo XML.
- escribir(): escribe un documento analizado en un archivo XML.
- mostrar(): muestra el contenido de un documento analizado.

5.3.5. ResultadoSinFormato

Esta clase mantiene toda la información relativa a un ítem (texto o frase) analizado sintácticamente en formato textual.

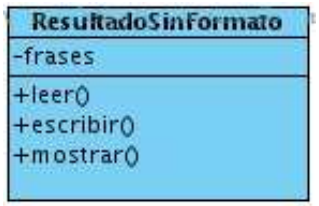

Figura 5.6: Clase DocumentoFC

Atributos

- frase: resultado obtenido por el análisis agrupado en frases.

Métodos

- leer(): lee un documento analizado desde un archivo de texto.
- escribir(): escribe un documento analizado en un archivo texto.
- mostrar(): muestra el contenido de un documento analizado.

5.3.6. ToResultado

Clase abstracta que contiene los métodos necesarios para crear resultados a partir de las salidas de distintos analizadores

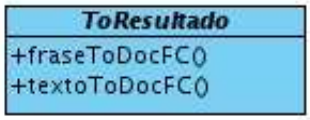

Figura 5.7: Clase ToResultado

Métodos

- fraseToResultadoFC(): Analiza una única frase y devuelve su resultado en el formato común.
- textoToResultadoFC(): Analiza un texto y devuelve sus resultados en el formato común.
- fraseToResultadoSF(): Analiza una única frase y devuelve su resultado en texto plano.
- textoToResultadoSF(): Analiza un texto y devuelve sus resultados en texto plano.

5.3.7. ToResultadoImp

Clase que implementa los métodos necesarios para crear resultados a partir de la salida de un analizador concreto.

Métodos

- fraseToResultadoFC(): Analiza una única frase y devuelve su resultado en el formato común.

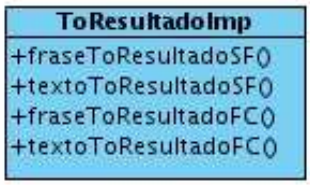

Figura 5.8: Clase ToResultado

- textoToResultadoFC(): Analiza un texto y devuelve sus resultados en el formato común.
- fraseToResultadoSF(): Analiza una única frase y devuelve su resultado en texto plano.
- textoToResultadoSF(): Analiza un texto y devuelve sus resultados en texto plano.

5.3.8. Item

Clase abstracta que representa a un objeto o ítem susceptible de ser analizado.

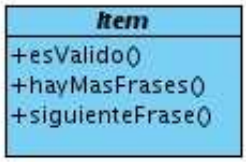

Figura 5.9: Clase Item

Métodos

- esValido(): Método que devuelve verdadero en caso de que el ítem pueda ser analizado, falso en caso contrario.
- hayMasFrases(): Método que devuelve verdadero si el item contiene frases que aún no se han analizado.
- siguienteFrase(): Devuelve la siguiente frase a analizar.

5.3.9. Frase

Clase que representa a una frase analizable.

Atributos

- frase: Conjunto de caracteres que forman la frase a analizar.

Figura 5.10: Clase Frase

Métodos

- esValido(): Método que devuelve verdadero en caso de que la frase pueda ser analizada, falso en caso contrario.
- getFrase(): Devuelve una cadena que contiene la frase a analizar.

5.3.10. Texto

Clase que representa a un texto analizable.

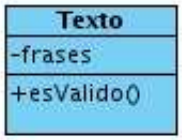

Figura 5.11: Clase Texto

- frase: Conjunto de frases que forman el texto a analizar.

Métodos

- esValido(): Método que devuelve verdadero en caso de que el texto pueda ser analizada, falso en caso contrario.

5.4. Casos de uso

En esta sección presentarán el conjunto de casos de uso que componen el sistema expuesto. En primer lugar, nos centraremos en el diagrama de casos de uso general de la aplicación y, posteriormente, iremos detallando cada uno de los casos de uso que lo componen.
A continuación se describirán los casos de uso detalladamente y se dividirán en otros más sencillos en caso de ser necesario.

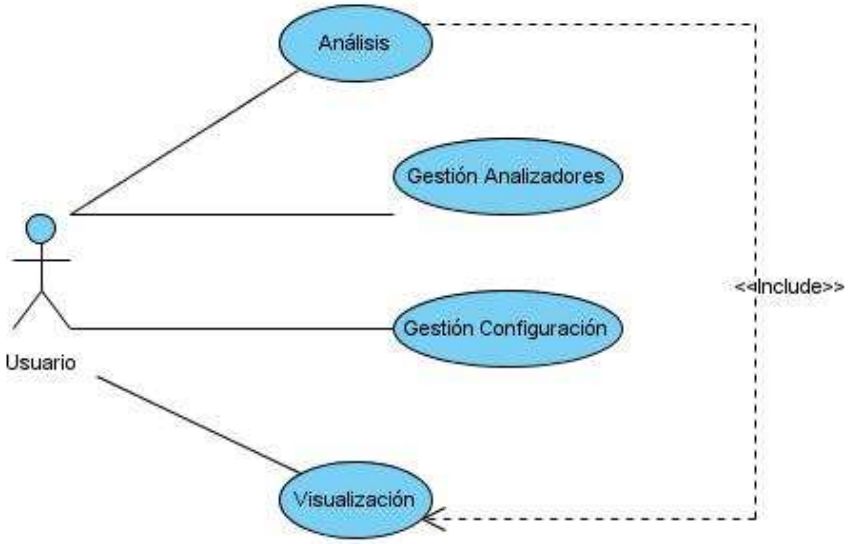

Figura 5.12: Diagrama de casos de uso general

5.4.1. Caso de uso: Gestión Configuración

Este caso de uso permite consultar y modificar parámetros importantes en el funcionamiento de la aplicación, como puede ser la estructura de directorios donde se almacenan los archivos necesarios.

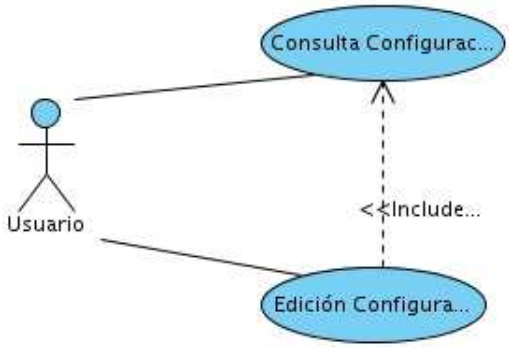

Figura 5.13: Diagrama de casos de uso Gestión configuración

Gestión Configuracion (Escenario 1): Consultar configuración

Descripción: en este escenario se describe cómo se pueden consultar parámetros importantes del sistema.

Flujo principal

1. El usuario solicita ver los datos relativos a la configuración de la aplicación.
2. Dichos datos son mostrados al usuario

Poscondiciones

- Se muestran al usuario los datos requeridos.

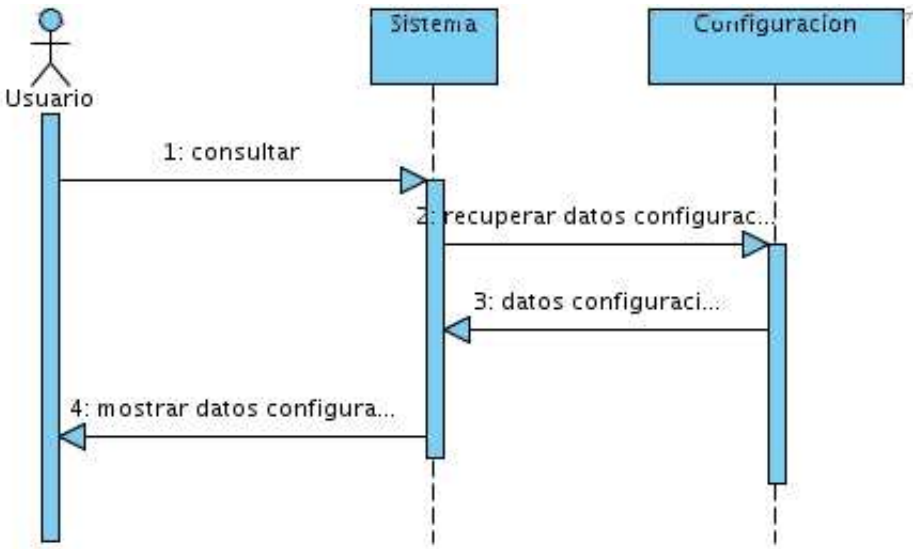

Figura 5.14: Diagrama de secuencia: Consultar configuración

Gestión Configuracion (Escenario 2): Editar configuración

Descripción: en este escenario se describe cómo se pueden modificar parámetros importantes del sistema.

Flujo principal

1. El usuario solicita ver los datos relativos a la configuración de la aplicación.
2. Dichos datos son mostrados al usuario
3. El usuario introduce los nuevos datos.
4. La configuración del sistema se actualiza con los nuevos datos introducidos por el usuario.

Flujo excepcional

- Se muestra un mensaje de error si los datos introducidos por el usuario no son correctos.

Poscondiciones

- La información de configuración es actualizada en el sistema.

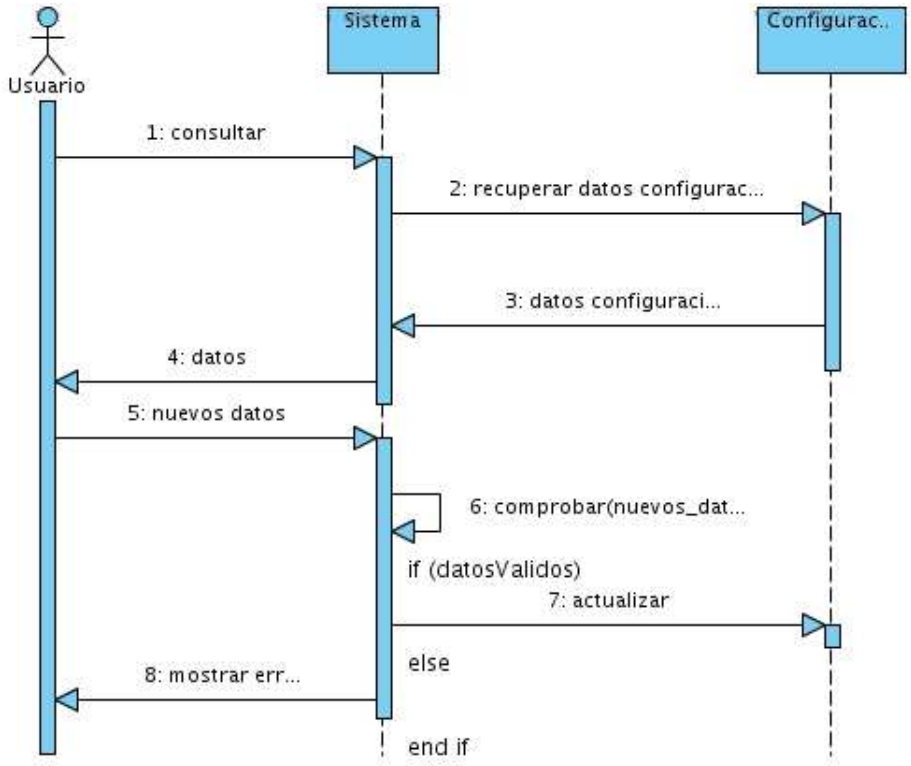

Figura 5.15: Diagrama de secuencia: Editar configuración

5.4.2. Caso de uso: Análisis sintáctico

El análisis sintáctico se puede llevar a cabo de dos maneras: a partir de una frase individual, o a partir de un documento (conjunto de frases). Una vez realizado el análisis el resultado será transformado al formato común propuesto en este proyecto. También es importante señalar que para poder llevarse a cabo esta tarea es imprescindible haber seleccionado previamente uno o varios analizadores sintácticos.

Análisis sintáctico (Escenario 1): Analizar frase

Descripción: en este escenario se describe cómo se analiza sintácticamente una frase.

Precondiciones

- Se ha seleccionado previamente uno o varios analizadores.

Flujo principal

1. El usuario introduce la frase a analizar.
2. El sistema envía la frase al analizador o analizadores correspondientes.
3. Se lanza el análisis y se obtiene el resultado en formato de texto plano.

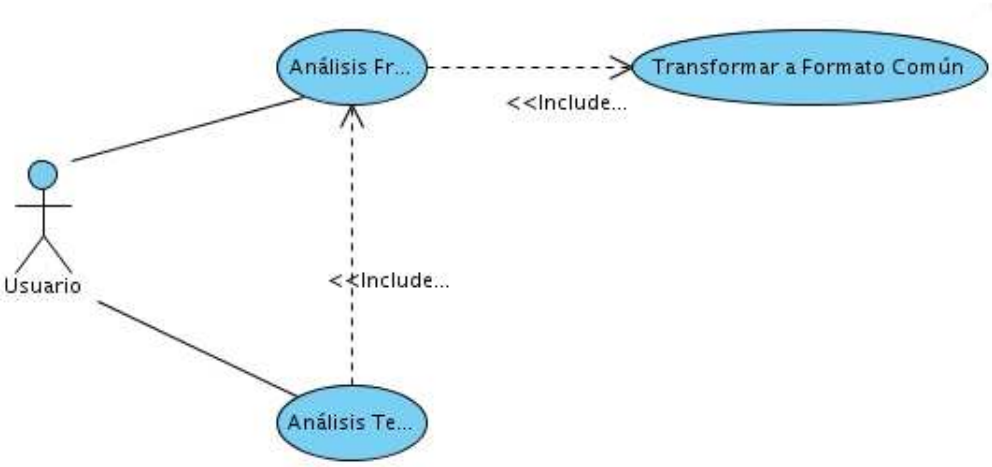

Figura 5.16: Diagrama de casos de uso Análisis sintáctico

Flujo excepcional

- Si la frase no es reconocida se muestra un mensaje de error.

Poscondiciones

- El resultado del análisis se obtiene como texto plano.

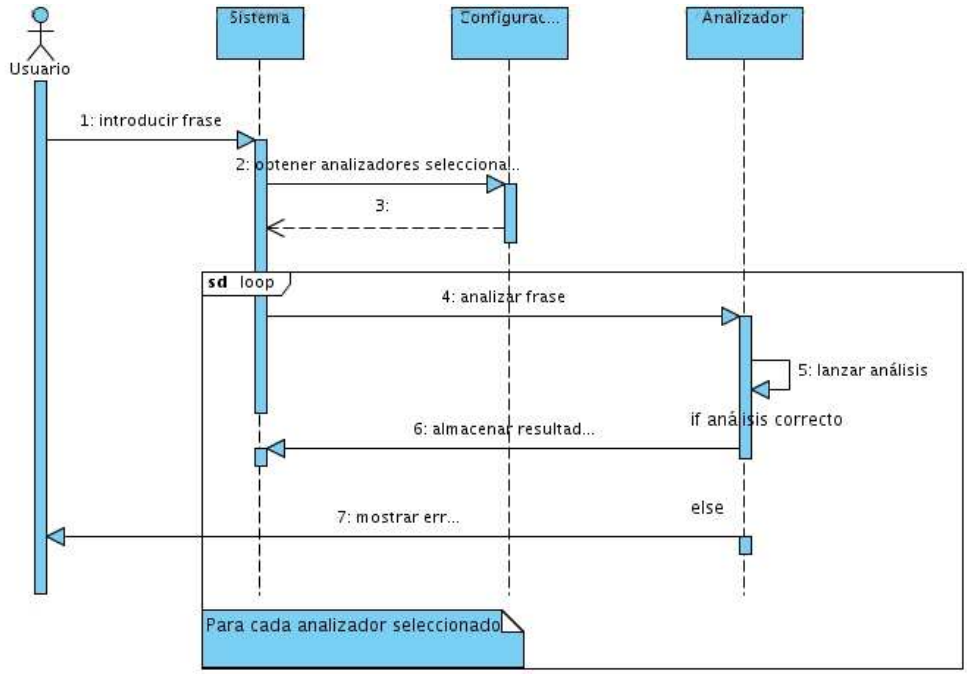

Figura 5.17: Diagrama de secuencia: Analizar frase

Análisis sintáctico (Escenario 2): Analizar texto

Descripción: en este escenario se describe cómo se analiza sintácticamente un texto compuesto por varias frases.

Precondiciones

- Han sido seleccionados previamente uno o varios analizadores.

Flujo principal

1. El usuario introduce la ruta del documento de texto a analizar.
2. El documento es dividido en frases.
3. El sistema envía cada frase al analizador o analizadores correspondientes.
4. Se lanza el análisis y se obtiene el resultado en formato de texto plano.

Flujo excepcional

- Si el documento no tiene un formato correcto se muestra un mensaje de error.
- Si una frase no es reconocida se muestra un mensaje de error.

Poscondiciones

- El resultado del análisis se obtiene como texto plano.

Análisis sintáctico (Escenario 3): Transformar a formato común

Descripción: en este escenario se describe transformar el resultado de un análisis sintáctico previo al formato propuesto en este proyecto.

Precondiciones

- Ha sido realizado previamente el análisis sintáctico de una frase o de un texto.

Flujo principal

1. El usuario solicita obtener el resultado en el formato común.
2. El sistema transforma los resultados obtenidos en el formato concreto de la aplicación al formato común.
3. El resultado en el formato común es mostrado al usuario.

Poscondiciones

- El resultado del análisis se obtiene en el formato común.

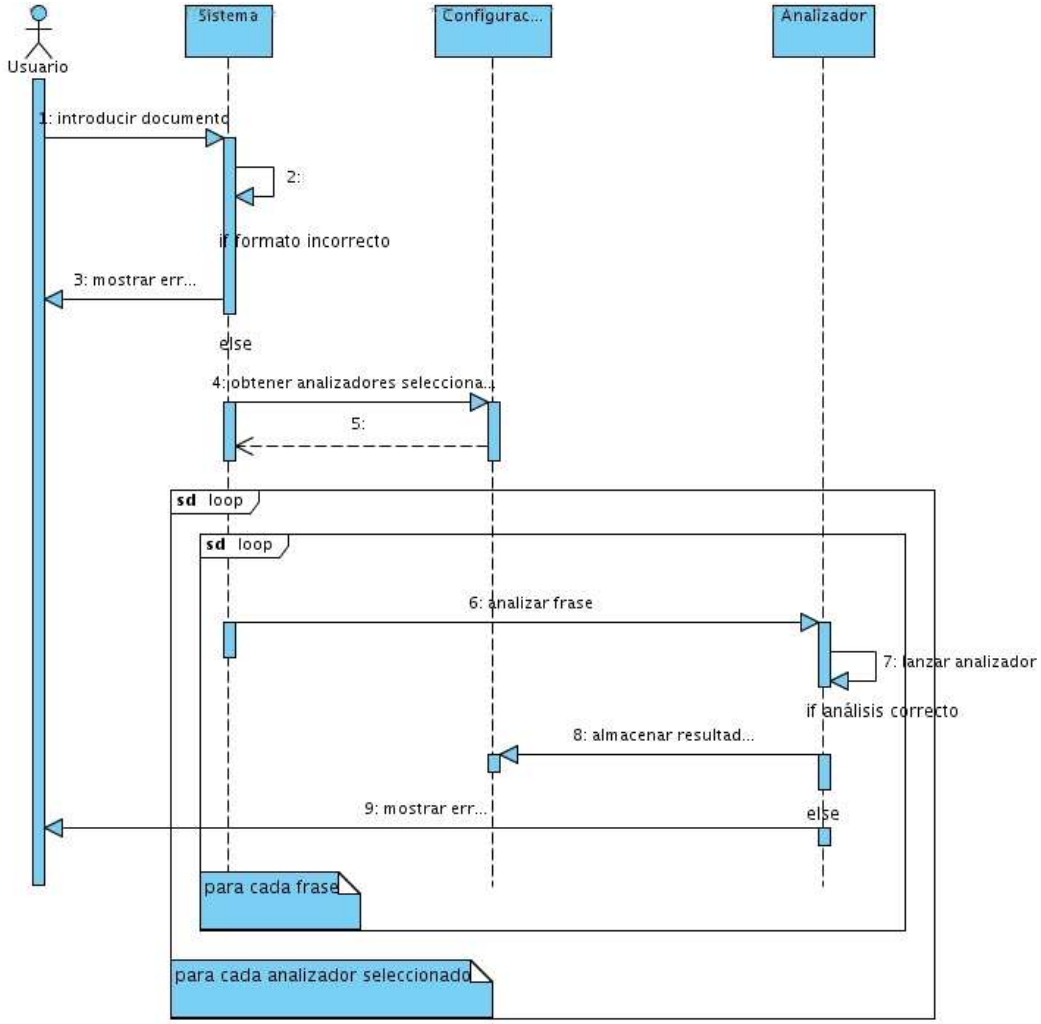

Figura 5.18: Diagrama de secuencia: Analizar texto

5.4.3. Caso de uso: Gestión Analizadores

Este caso de uso permite realizar una completa gestión de los analizadores sintácticos presentes en el sistema. Esto es, crear un nuevo analizador, borrar, modificar o consultar uno existente y, por último, seleccionar aquellos que se quieren utilizar a la hora de lanzar un análisis sintáctico.

Gestión Analizadores (Escenario 1): Alta analizador sintáctico

Descripción: en este escenario se describe cómo se puede añadir un nuevo analizador sintáctico al sistema.

Flujo principal

1. El usuario introduce el nombre del nuevo analizador.

2. Si el nombre introducido no corresponde con el de otro analizador existente, se introducen todos los parámetros necesarios.

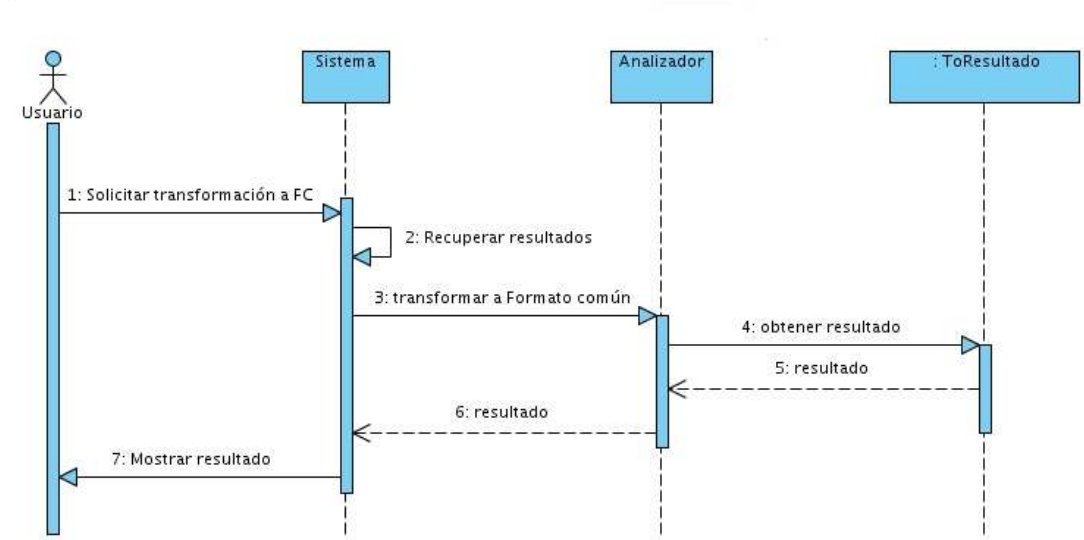

Figura 5.19: Diagrama de secuencia: Transformar a formato común

3. Si los datos introducidos son correctos, el nuevo analizador será almacenado en el sistema.

Flujo excepcional

- Si los datos introducidos no cumplen el formato correspondiente, o falta algún parámetro obligatorio por cubrir, se mostrará un mensaje de error.

Poscondiciones

- El nuevo analizador es añadido al sistema, y pude ser utilizado para futuros análisis.

Gestión Analizadores (Escenario 2): Baja analizador sintáctico

Descripción: en este escenario se describe cómo se puede eliminar un analizador sintáctico del sistema.

Precondiciones

- El analizador a eliminar debe existir en el sistema.

Flujo principal

1. El usuario solicita una lista de los analizadores disponibles.
2. El sistema muestra dicha lista y el usuario selecciona el analizador a eliminar.
3. El sistema pide confirmación para borrar el analizador.
4. El usuario confirma y el analizador sintáctico es eliminado del sistema.

Flujo excepcional

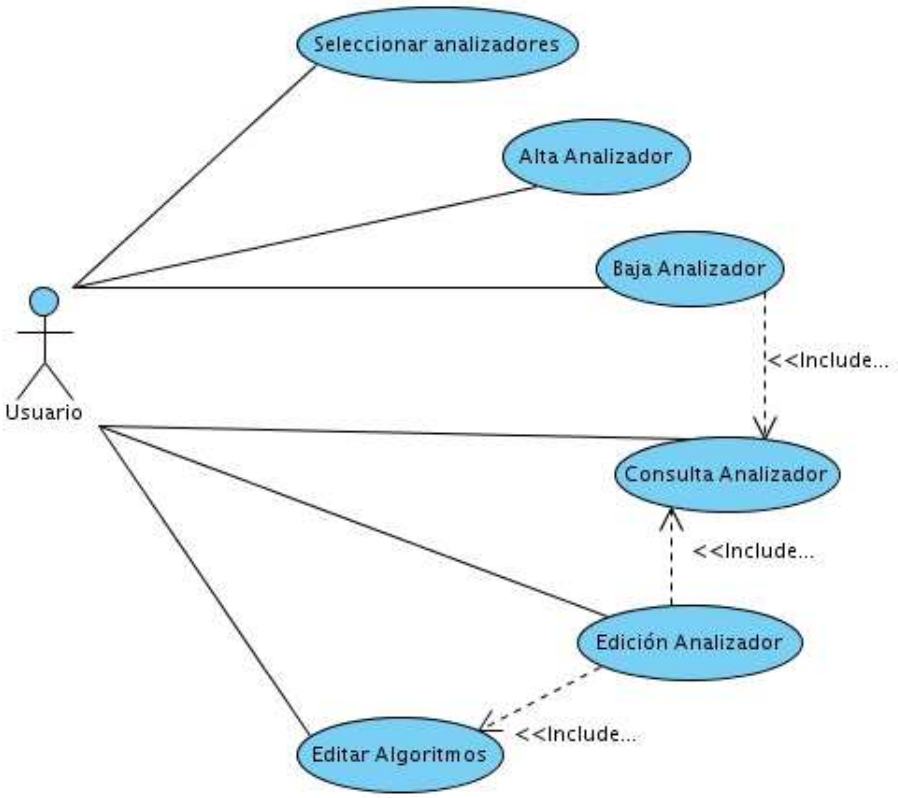

Figura 5.20: Diagrama de casos de uso: Gestión analizadores

- El usuario puede abandonar el proceso en cualquier momento.

Poscondiciones

- El analizador es eliminado del sistema, y no podrá ser utilizado para futuros análisis.

Gestión Analizadores (Escenario 3): Modificación analizador sintáctico

Descripción: en este escenario se describe cómo se puede modificar un analizador sintáctico presente en el sistema.

Precondiciones

- El analizador a eliminar debe existir en el sistema.

Flujo principal

1. El usuario solicita una lista de los analizadores disponibles.
2. El sistema muestra dicha lista y el usuario selecciona el analizador a modificar.
3. El usuario introduce los nuevos datos.

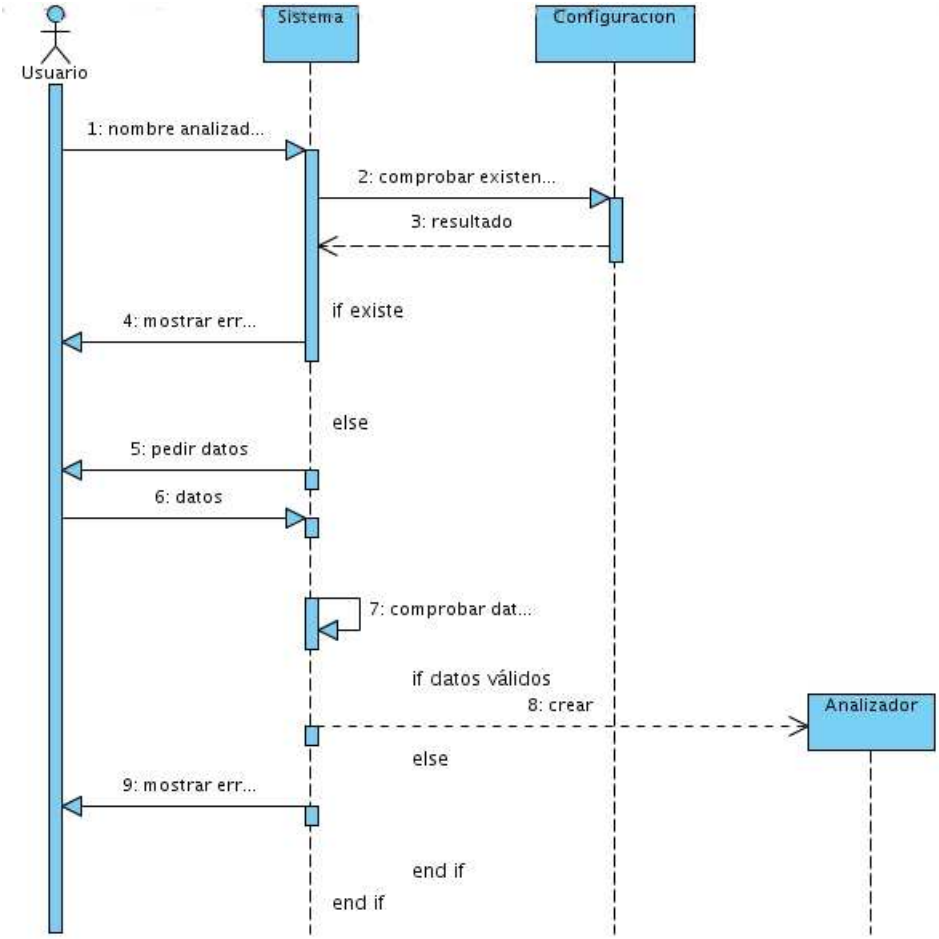

Figura 5.21: Diagrama de secuencia: Alta analizador

4. Si los datos introducidos son correctos, la información del analizador sintáctico en el sistema se actualiza.

Flujo excepcional

- Si los datos introducidos no son correctos, se muestra un mensaje de error.

Poscondiciones

- Los datos del analizador son actualizados.

Gestión Analizadores (Escenario 4): Consulta analizador sintáctico

Descripción: en este escenario se describe cómo se consulta un analizador sintáctico presente en el sistema.

Precondiciones

- El analizador a consultar debe existir en el sistema.

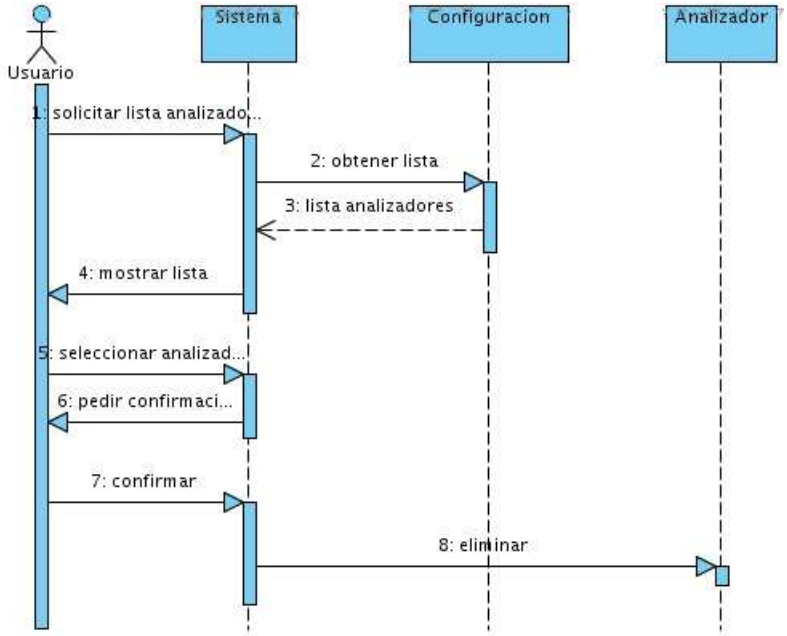

Figura 5.22: Diagrama de secuencia: Baja analizador

Flujo principal

1. El usuario solicita una lista de los analizadores disponibles.
2. El sistema muestra dicha lista y el usuario selecciona el analizador a consultar.
3. El sistema muestra los datos del analizador sintáctico.

Poscondiciones

- Los datos del analizador sintáctico seleccionado son mostrados al usuario.

Gestión Analizadores (Escenario 5): Seleccionar Analizadores Sintácticos

Descripción: en este escenario se describe cómo se selecciona la lista de analizadores sintácticos activos.

Precondiciones

- Los analizadores a seleccionar deben existir en el sistema.

Flujo principal

1. El usuario solicita una lista de los analizadores disponibles.
2. El sistema muestra dicha lista y el usuario selecciona los analizadores que crea conveniente.
3. El sistema establece los analizadores seleccionados como activos.

Poscondiciones

- La proxima vez que el usuario desee realizar un análisis sintáctico, se emplearán los analizadores seleccionados

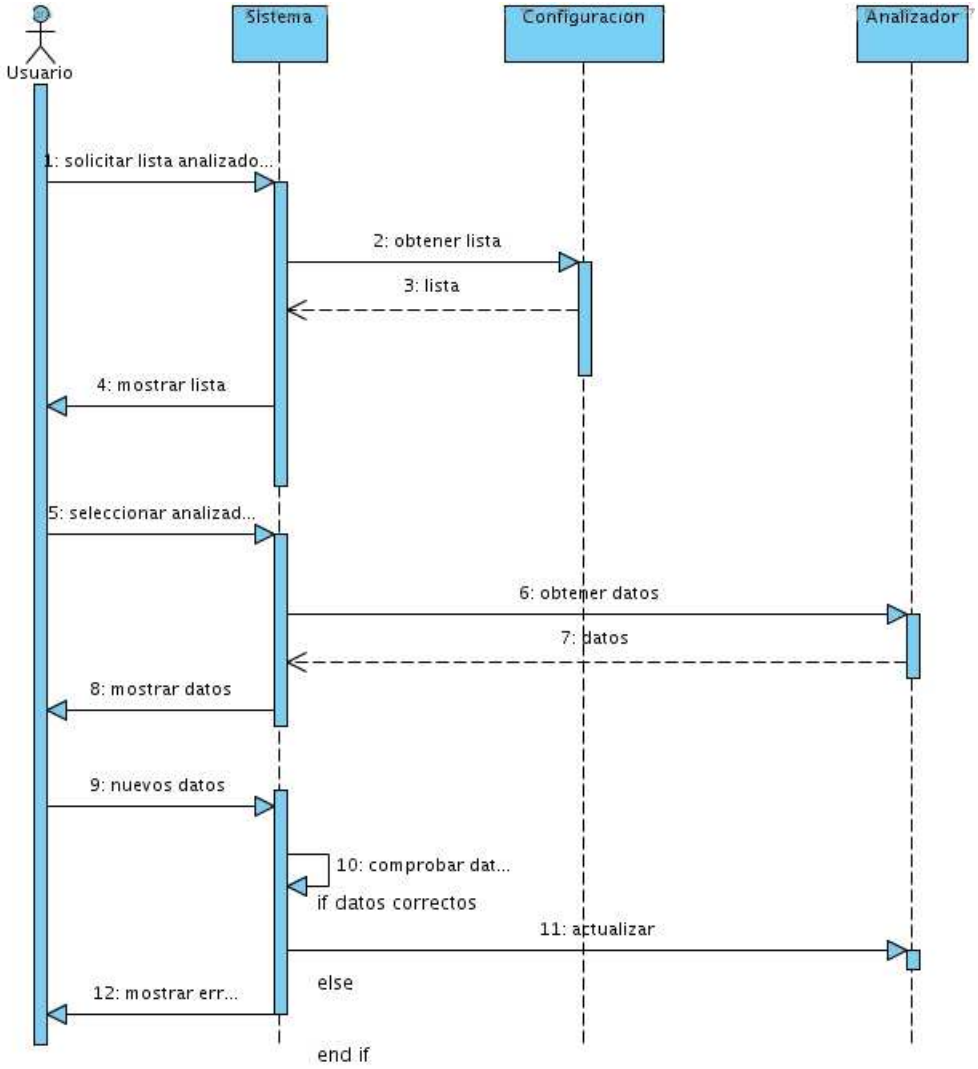

Figura 5.23: Diagrama de secuencia: Modificación analizador

Gestión Analizadores (Escenario 6): Editar algoritmos

Descripción: en este escenario se describe cómo se editan los algoritmos de obtención de resultados del anállisis.

Precondiciones

- El analizador debe existir en el sistema.

Flujo principal

1. El usuario solicita una lista de los analizadores disponibles.
2. El sistema muestra dicha lista y el usuario selecciona el analizador sintáctico cuyos algoritmos desee modificar.
3. El usuario modifica los algoritmos a su antojo.

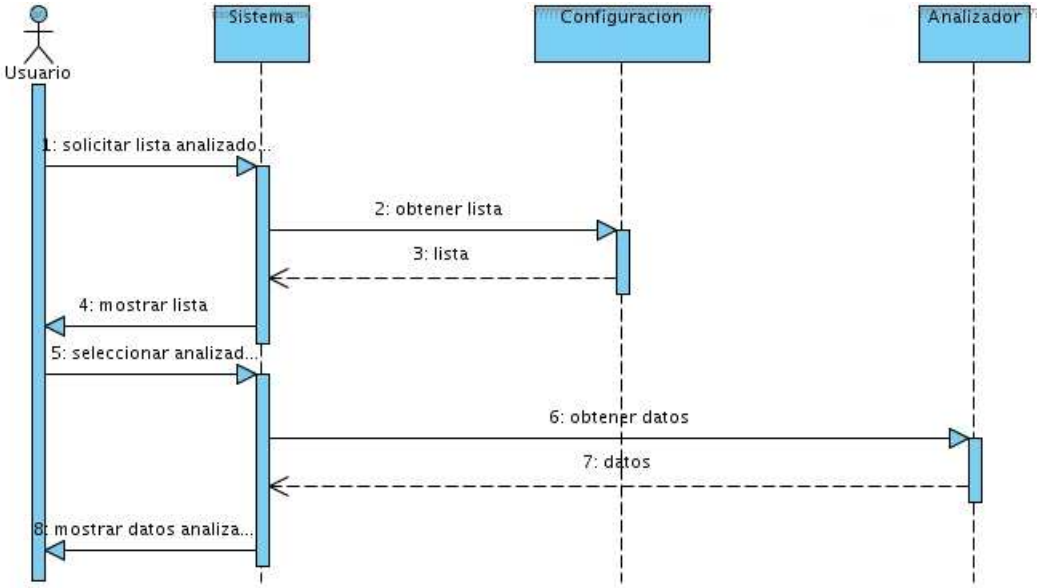

Figura 5.24: Diagrama de secuencia: Consulta analizador

4. El sistema compila los algoritmos introducidos.

5. Si la compilación es correcta los algoritmos son actualizados en el sistema.

Flujo excepcional

- Si la compilación no es correcta, se mostrará un mensaje de error.

Poscondiciones

- Los algoritmos se han modificado.

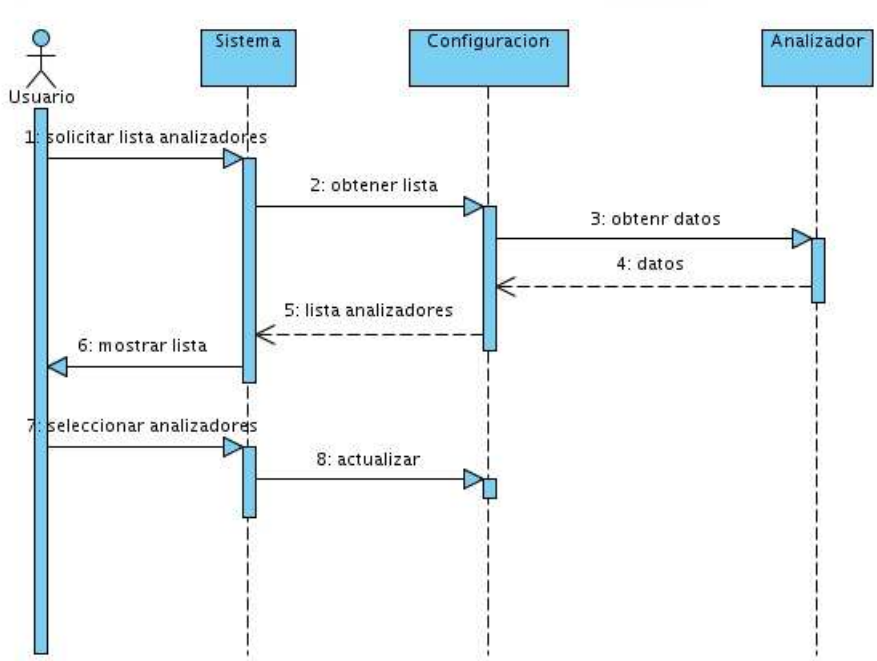

Figura 5.25: Diagrama de secuencia: Seleccionar analizadores

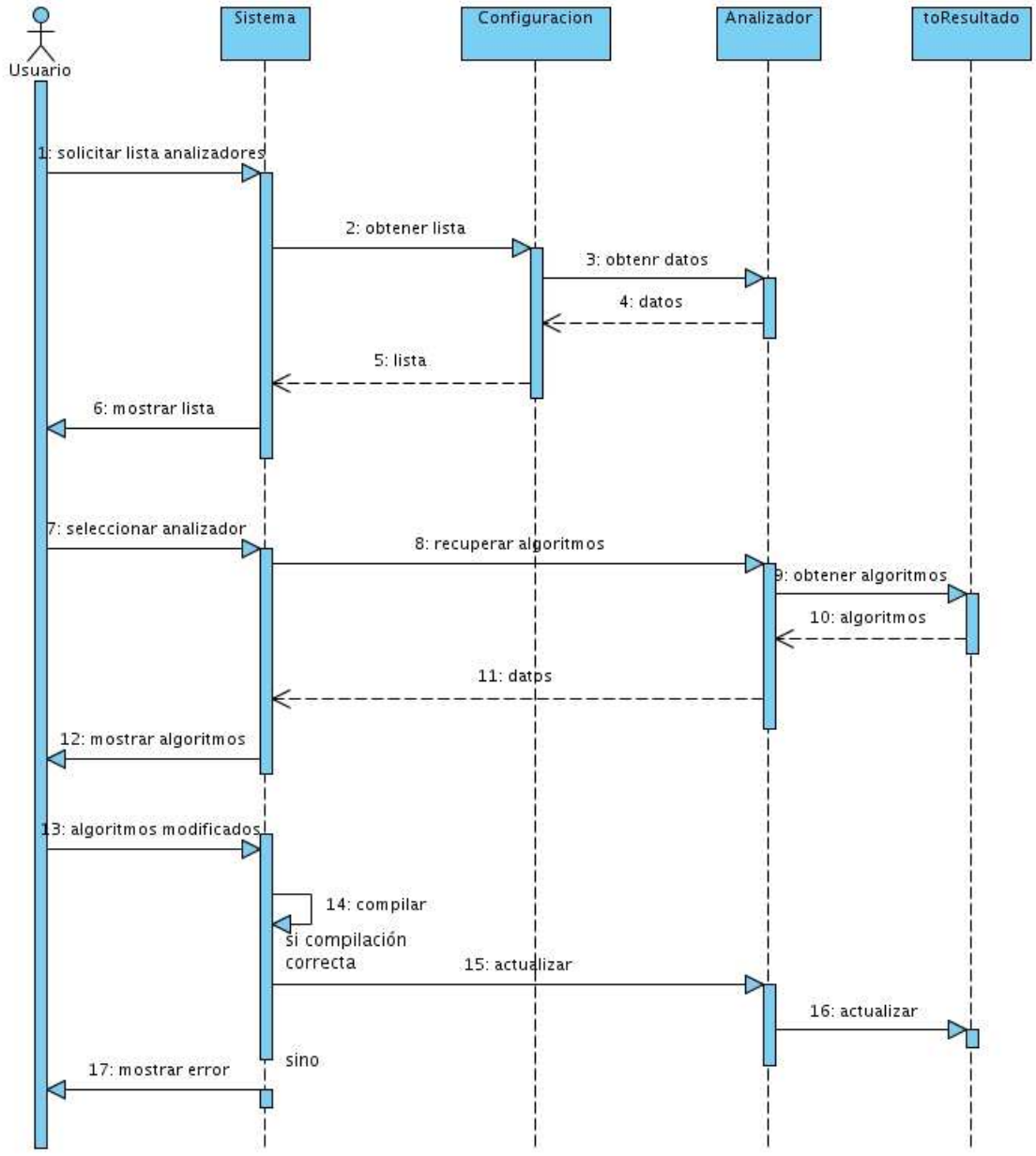

Figura 5.26: Diagrama de secuencia: Editar algoritmos

Capítulo 6

Diseño

En este capítulo se detalla el diseño del sistema desarrollado en este proyecto. El objetivo de esta fase es especificar cómo se resuelve el problema descrito en el análisis de requisitos (5).

6.1. Especificación de las clases

En este apartado del Manual Técnico se pasa a detallar las clases generadas durante el proceso de diseño del sistema. Se ha optado por presentar las mismas encapsuladas en los paquetes que las contienen.

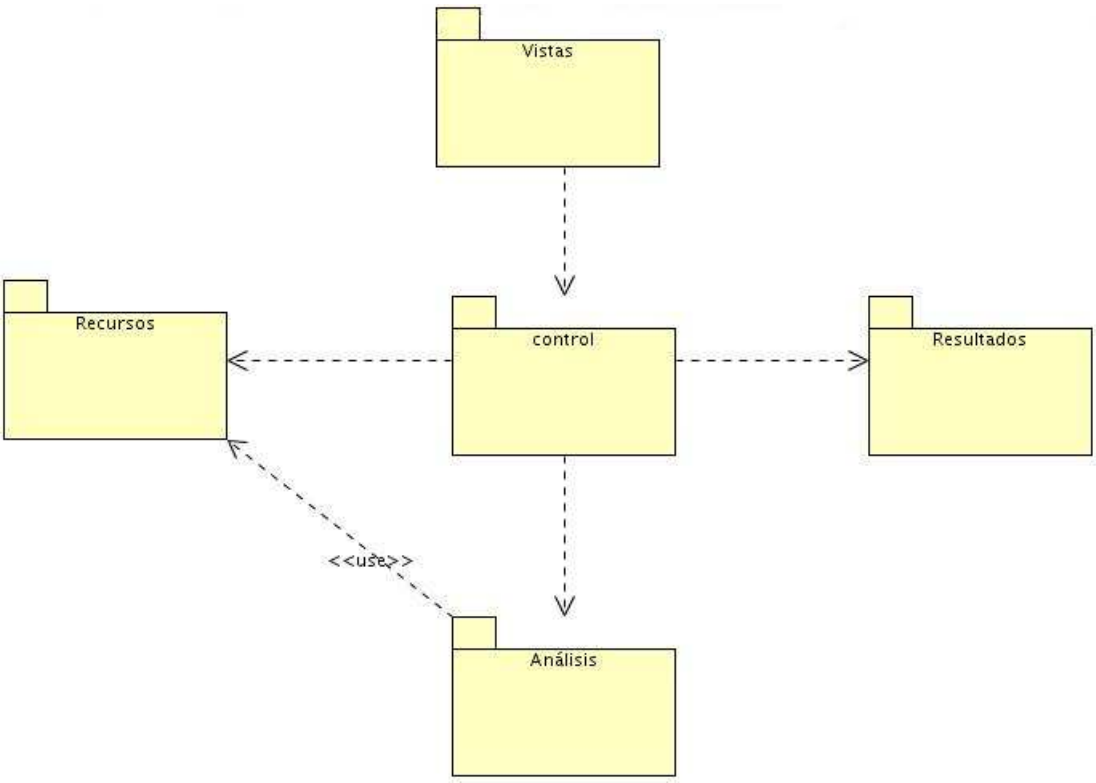

Figura 6.1: Diagrama de clases general

6.1.1. Paquete Análisis

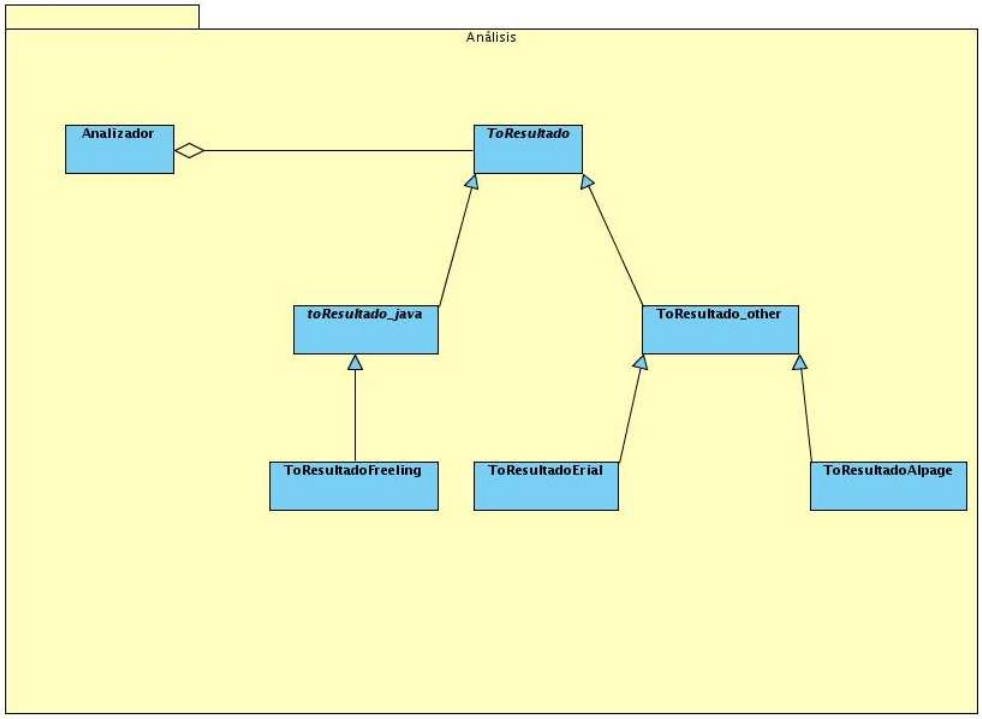

Figura 6.2: Paquete Análisis

Clase Analizador

Descripción

Esta clase implementa la funcionalidad correspondiente a un analizador sintáctico.

Atributos

- nombre: nombre único del analizador sintáctico que lo identifica y diferencia de los demás analizadores.
- directorio_base: directorio base del analizador.
- fichero_conf: fichero XML de configuración del analizador.
- propiedades: características generales que definen el comportamiento de un analizador
- clase_algoritmos: clase que contiene la implementación de los algoritmos para analizar un ítem y pasarlo a formato común para este analizador en concreto.
- script: ruta del ejecutable que lanza el analizador sintáctico.

Métodos

- actualizar(): actualiza las características del analizador.

- borrar(): elimina un analizador existente del sistema.
- get(): devuelve una propiedad del analizador.
- set(): establece/modifica una propiedad del analizador.
- analizarFrase(): analiza sintácticamente una frase.
- analizarDocumento(): analiza sintácticamente un documento.
- getClaseAlgoritmos(): devuelve una instancia de la clase que contiene los algoritmos concretos del analizador.

Clase ToResultado

Clase abstracta que contiene los métodos necesarios para crear resultados a partir de las salidas de distintos analizadores.

Métodos

- fraseToResultadoFC(): Analiza una única frase y devuelve su resultado en el formato común.
- textoToResultadoFC(): Analiza un texto y devuelve sus resultados en el formato común.
- fraseToResultadoSF(): Analiza una única frase y devuelve su resultado en texto plano.
- textoToResultadoSF(): Analiza un texto y devuelve sus resultados en texto plano.
- toString(): Devuelve en formato de texto el código fuente de la clase;
- getTipo(): Devuelve el tipo de analizador sintáctico al que se refiere esta clase. Puede ser "java" u "otro".

Clase ToResultado_java

Clase abstracta que contiene los métodos necesarios para crear resultados a partir de las salidas de distintos analizadores en *Java*, implementando todas las funcionalidades comunes para los mismos.

Métodos

- analizarFrase(): Analiza sintácticamente una frase dada. Debe ser rescrito por los distintos analizadores sintácticos escritos en Java.
- toResultadoFC(): Dada la salida del analizador sintáctico, la transforma al formato común propuesto en este proyecto.

Clase ToResultado_other

Clase abstracta que contiene los métodos necesarios para crear resultados a partir de las salidas de analizadores que no están escritos en java y que, por lo tanto, se tienen que ejecutar como procesos independientes de la aplicación.

Atributos

- proceso: Proceso que se va a utilizar para lanzar el analizador sintáctico.

Métodos

- buildCommand(): Método abstracto que construye el comando que se tiene que ejecutar para lanzar el análisis. Deberá ser implementado para cada clase derivada de ésta.
- lanzarAnalizador(): Método que, utilizando el comando construído por el método *buildCommand()* lanza el análisis sintáctico como un proceso independiente y recibe los resultados en formato textual.

Clase ErialToResultado

Clase que implementa los algoritmos necesarios para llevar a cabo el análisis sintáctico y la conversión al formato común para el caso concreto del analizador *Erial*. Hereda de la clase ToResultado_other, por lo que sólo implementa los métodos no implementados por ésta. Esta clase puede ser modificada por el usuario.

Atributos

- deps: Contiene las dependencias detectadas entre los distintos componentes de la frase por el analizador sintáctico.
- etiqs: Contiene las etiquetas de los distintos componentes detectadas por el analizador sintáctico.
- fraseAnalizada: Contiene la frase original a analizar.
- dirBase: Directorio base de la aplicación, donde se encuentra el script a ejecutar.
- SCRIPT_ERIAL: Constante que contiene la ruta del script que lanza el analizador.

Métodos

- toResultadoFC(): Método que transforma la salida estándar del analizador *Erial* al formato común propuesto en este proyecto.
- buildCommand(): Método que construye el comando que se tiene que ejecutar para lanzar el análisis sintáctico con *Erial*.
- getDirAplicacion: Método que devuelve el directorio base de la aplicación donde se encuentra el script que lanza el analizador sintáctico *Erial*.

Clase AlpageToResultado

Clase que implementa los algoritmos necesarios para llevar a cabo el análisis sintáctico y la conversión al formato común para el caso concreto del analizador *Alpage*. Hereda de la clase ToResultado_other, por lo que sólo implementa los métodos no implementados por ésta. Recordemos que esta clase puede ser modificada por el usuario.

Métodos

- toResultadoFC(): Método que transforma la salida estándar del analizador sintáctico *Alpage* al formato común propuesto en este proyecto.
- buildCommand(): Método que construye el comando que se tiene que ejecutar para lanzar el análisis sintáctico con *Alpage*.
- getDirAplicacion: Método que devuelve el directorio base de la aplicación donde se encuentra el script que lanza el analizador sintáctico *Alpage*.

6.1.2. Paquete Resultados

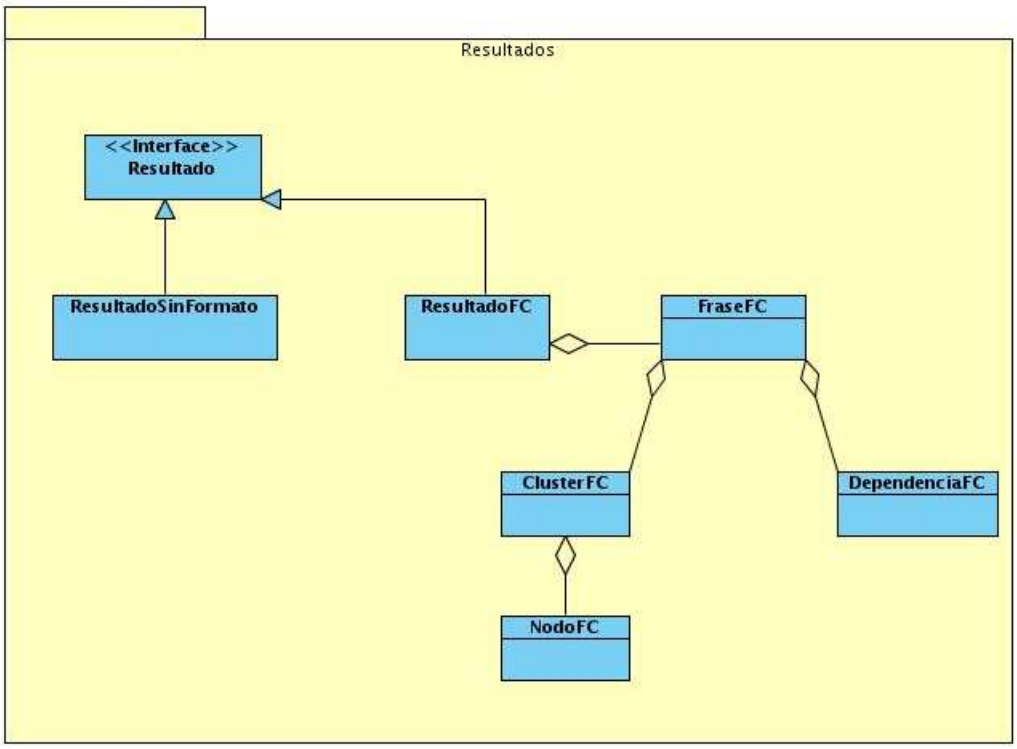

Figura 6.3: Paquete Resultados

Interfaz Resultado

Esta interfaz mantiene toda la información relativa a un ítem (texto o frase) analizado sintácticamente.

Métodos

- generarVistaTextual(): Método que genera una vista textual para el resultado.
- generarVistaGrafica(): Método que genera una vista gráfica para el resultado. En caso de que la clase derivada no tenga una implementación específica para este método, se mostrará un gráfico vacío.
- guardar(): Método que almacena el resultado del análisis sintáctico en un fichero.

Clase ResultadoFC

Clase que implementa la interfaz *Resultado* y que almacena el resultado de un análisis sintáctico en el formato propuesto en este proyecto.

Atributos

- frase: Frase analizada en el formato común propuesto.

Métodos

- generarVistaTextual(): Método que genera una vista textual para el resultado, que será mostrado en formato XML como se ha descrito en el apartado 1.5.1.
- generarVistaGrafica(): Método que genera una vista gráfica para el resultado.
- guardar(): Método que almacena el resultado del análisis sintáctico en un fichero XML.

Clase ResultadoSinFormato

Clase que implementa la interfaz *Resultado* y que almacena el resultado de un análisis sintáctico en el formato propio del analizador concreto empleado.

Atributos

- lineas: Almacena la salida original del analizador sintáctico, línea a línea.

Métodos

- generarVistaTextual(): Método que genera una vista textual para el resultado, que será mostrado en formato de texto.
- guardar(): Método que almacena el resultado del análisis sintáctico en un fichero de texto.

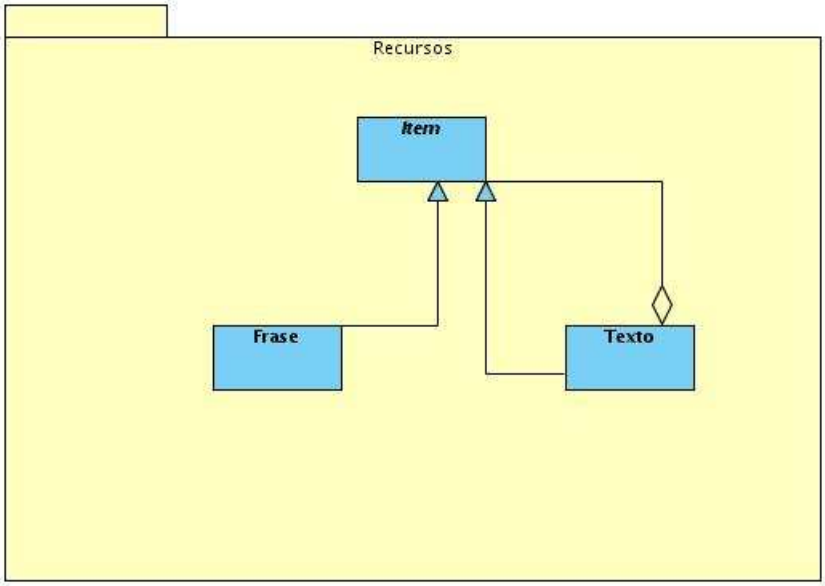

Figura 6.4: Paquete recursos

6.1.3. Paquete Recursos

Interfaz Item

Esta interfaz refleja el comportamiento esperado para un ítem que puede ser analizado, ya sea una frase o un texto compuesto por varias frases.

Métodos

- esValido(): Método que devuelve verdadero si el ítem es válido para realizar un análisis sintáctico sobre el o falso en caso contrario.
- analizar(): Método que devuelve el resultado del análisis sintáctico del ítem, realizado con un analizador determinado.

Clase Frase

Clase que implementa la interfaz *Item* y representa a una frase susceptible de ser analizada.

Atributos

- laFrase: Cadena que contiene la frase que se quiere analizar.

Métodos

- esValido(): Método que devuelve verdadero si la frase es válida para ser analizada, es decir, si no contiene algún carácter prohibido.
- analizar(): Método que devuelve el resultado del análisis sintáctico de la frase, realizado con una analizador sintáctico determinado.

Clase Texto

Clase que implementa la interfaz *Item* y representa a un texto susceptible de ser analizado.

Atributos

- frases: Secuencia de frases que forman el texto.

Métodos

- esValido(): Método que devuelve verdadero si el texto es válido para ser analizado, es decir, si todas sus frases son válidas.
- analizar(): Método que devuelve los resultados del análisis sintáctico del texto, realizado con una analizador sintáctico determinado.

6.1.4. Paquete Control

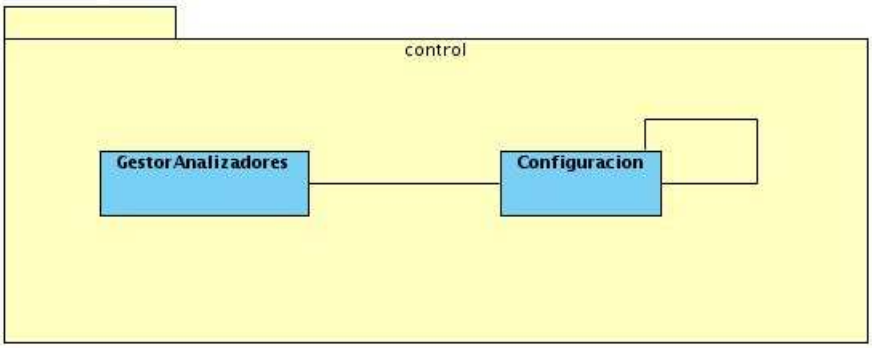

Figura 6.5: Paquete Control

Clase Configuración

Esta clase maneja toda la información necesaria para el funcionamiento de la aplicación, permitiendo configurarla para obtener el comportamiento deseado.

Atributos

- INSTANCIA: Instancia única de la clase configuración. Se usa el patrón *singleton* para que no se puedan crear varios objetos de esta clase y facilitar el acceso al objeto único.
- FICHERO_CONF: Ruta relativa del fichero de configuración de la aplicación.
- gestorAnalizadores: Objeto encargado de manejar toda la información referente a los analizadores sintácticos e interactuar con ellos.
- dir_base: Ruta relativa del directorio base de la aplicación.
- propiedades: Propiedades básicas de configuración de la aplicación.

Métodos

- getInstance(): Método que devuelve la instancia única de la clase (patrón *singleton*), creándola en caso de que sea necesario.
- createInstance(). Método privado que crea una instancia de esta clase.
- getPropiedades(): Método que devuelve las propiedades de configuración de la aplicación.
- actualizar(): Método que modifica las popiedades de configuración de la aplicación.
- cargar(): Carga los datos de configuración del fichero correspondiente.
- almacenar(): Almacena los datos de configuración en el fichero correspondiente.
- setPorDerfecto(): Asigna a la configuración los valores por defecto.
- getGestorAnalizadores(): Devuelve el valor del atributo gestorAnalizadores.

Clase GestorAnalizadores

Clase encargada de manejar toda la información relativa a los analizadores sintácticos y de comunicarse con ellos para llevar a cabo los análisis sintácticos.

Atributos

- analizadores: Lista que contiene a todos los analizadores sintácticos existentes en el sistema.
- actuales: Lista que contiene a los analizadores que han sido seleccionados para realizar el próximo análisis sintáctico.

Métodos

- crear(): Método que crea un nuevo analizador dadas sus características principales.
- borrar(): Método que, dado un nombre (que identifica de manera única a un analizador), elimina ese analizador del sistema, borrando sus datos del fichero de configuración, la clase que contiene sus algoritmos y el fichero de configuración propio del analizador.
- actualizar(): Método que modifica las características propias de un analizador, cambiándolas por unas características nuevas que recibe como parámetros.
- existe(): Método que, dado el nombre de un analizador sintáctico, devuelve cierto si existe un analizador con ese nombre y falso en caso contrario.
- getAnalizador(): Método que, dado el nombre de un analizador sintáctico, devuelve, si existe, el analizador sintáctico que responde a ese nombre.

- getAnalizadores(): Método que devuelve una lista con todos los analizadores sintácticos existentes.
- seleccionar(): Método que, dada una lista de nombres de analizadores, añade los analizadores correspondientes, si existen, a la lista de analizadores seleccionados, sobre los que se realizarán los próximos análisis.
- getSeleccionados(): Devuelve la lista de los analizadores que han sido seleccionados para realizar el próximo análisis sintáctico.

6.2. Diagramas de secuencia del diseño

En esta sección se presentan los diagramas de secuencia detallados que se corresponden a los casos de uso y diagramas de secuencia desarrollados durante el análisis.

6.2.1. Caso de uso: Gestión Configuración

Gestión Configuracion (Escenario 1): Consultar configuración

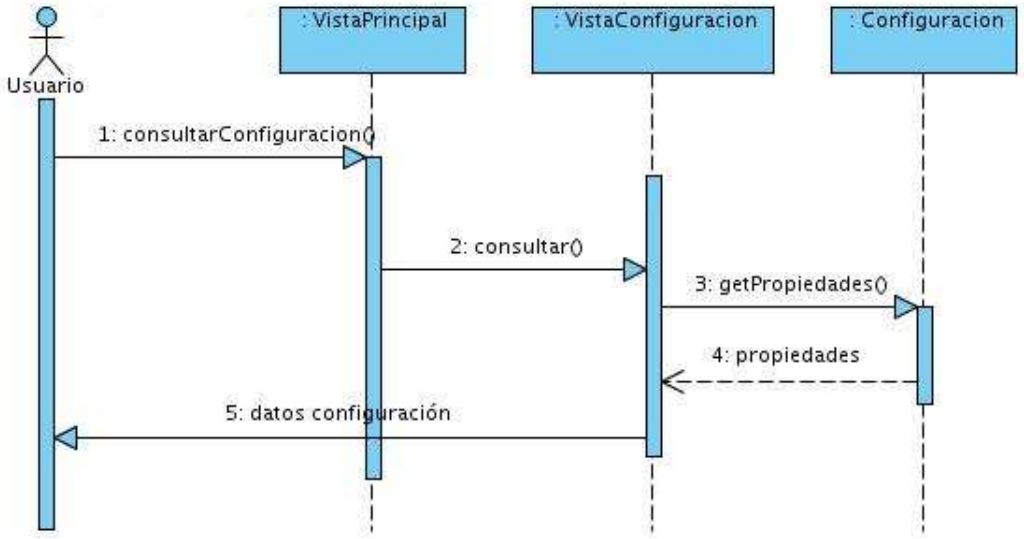

Figura 6.6: Diagrama de secuencia de diseño: Consultar configuración

Gestión Configuracion (Escenario 2): Editar configuración

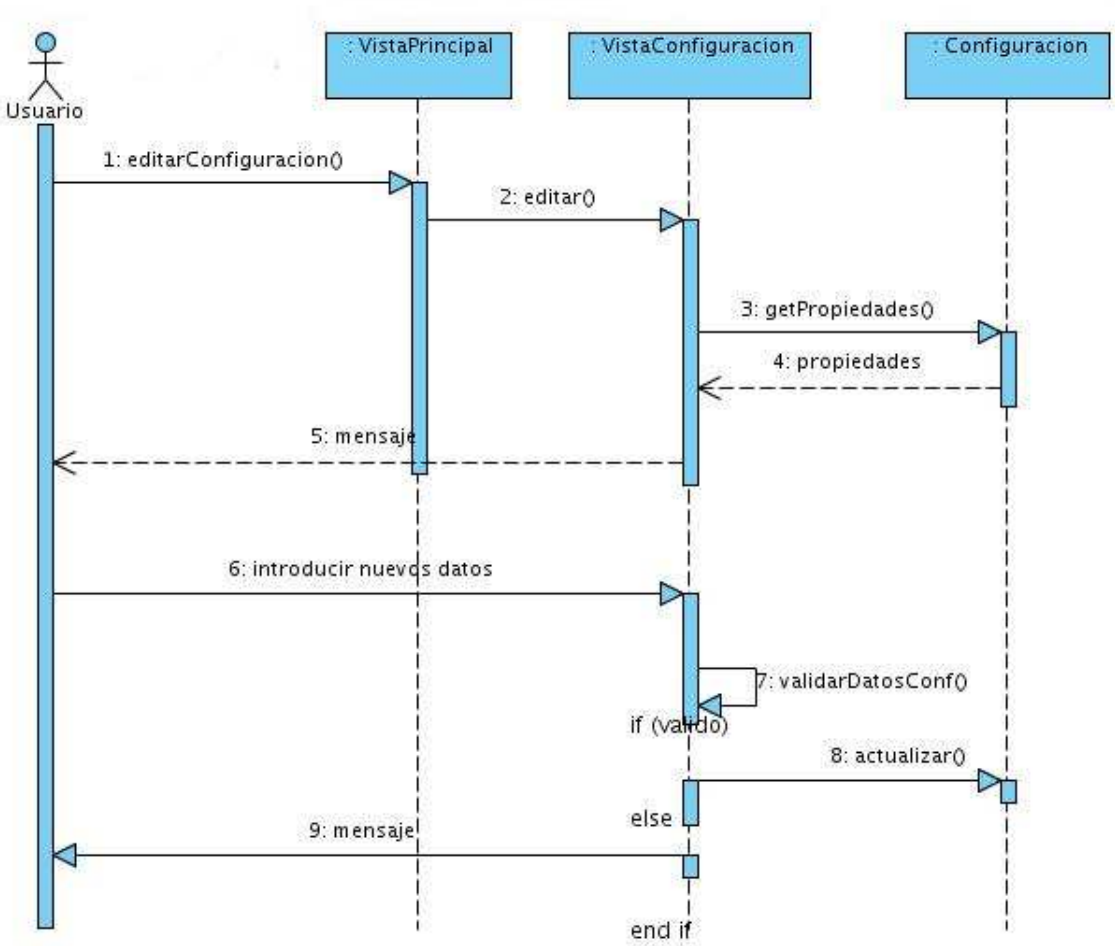

Figura 6.7: Diagrama de secuencia de diseño: Editar configuración

6.2.2. Caso de uso: Análisis sintáctico

Análisis sintáctico (Escenario 1): Analizar frase

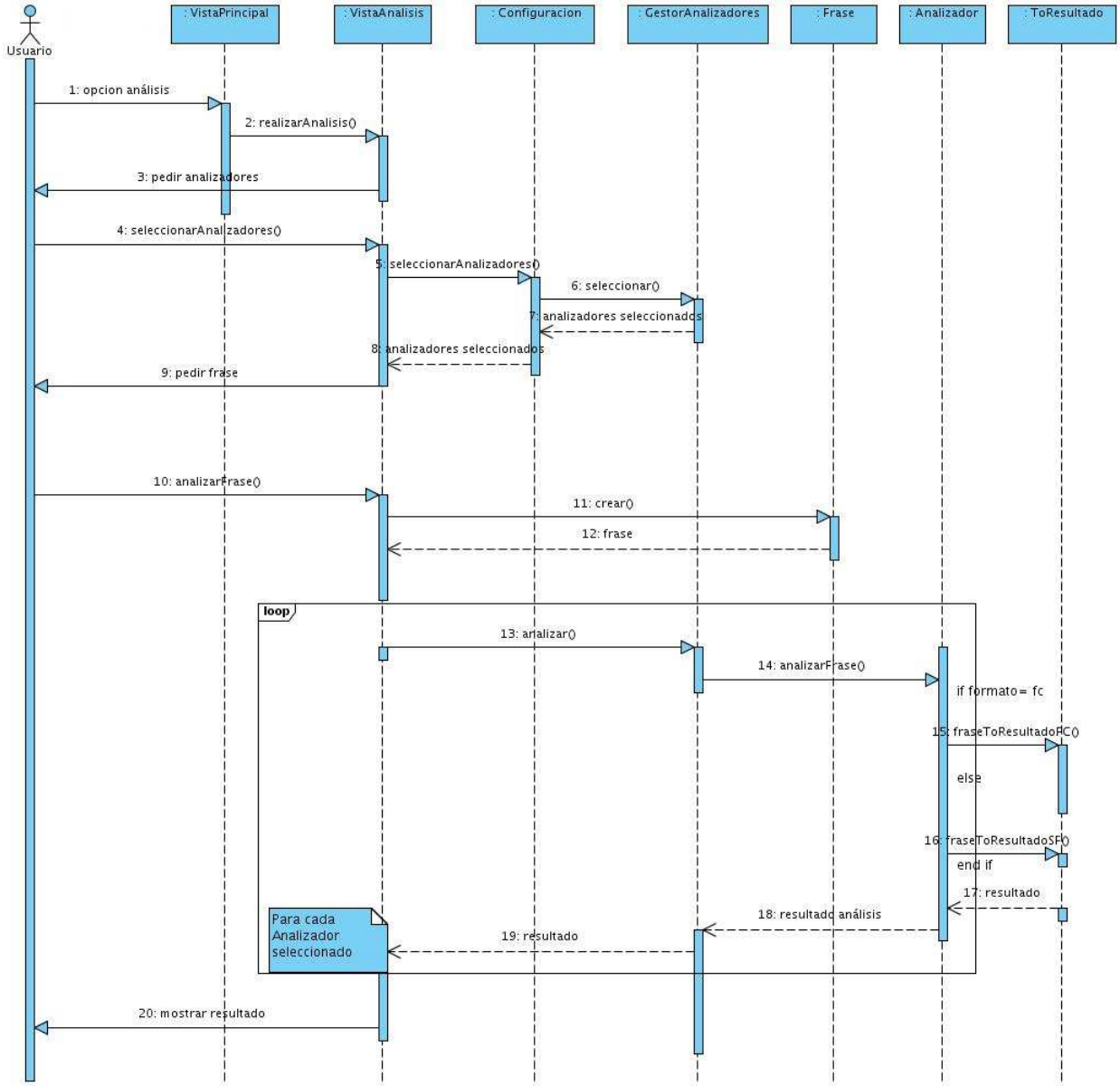

Figura 6.8: Diagrama de secuencia de diseño: Analizar frase

Análisis sintáctico (Escenario 2): Analizar texto

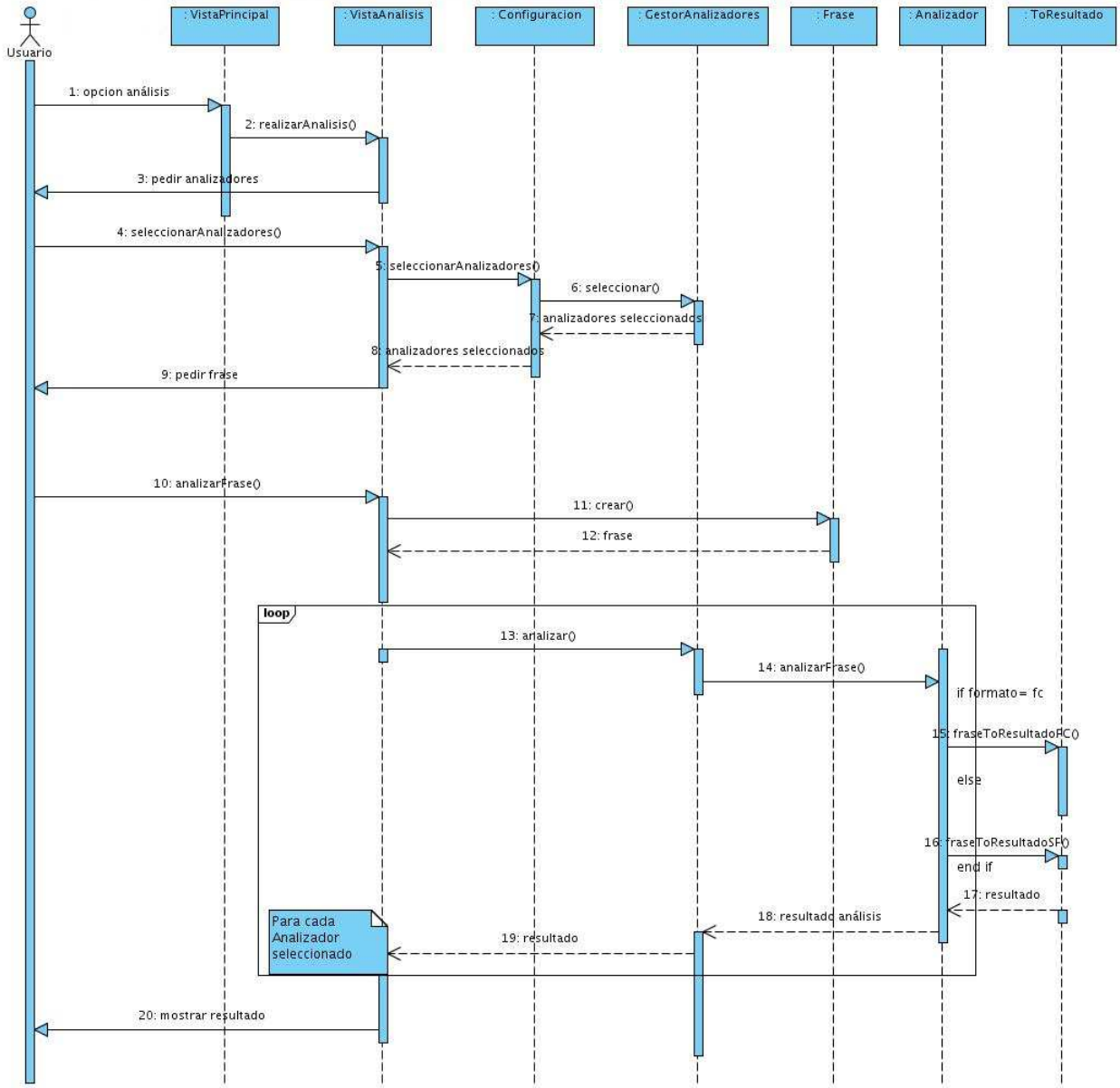

Figura 6.9: Diagrama de secuencia de diseño: analizar texto

6.2.3. Caso de uso: Gestión Analizadores

Gestión analizadores (Escenario 1): Alta analizador

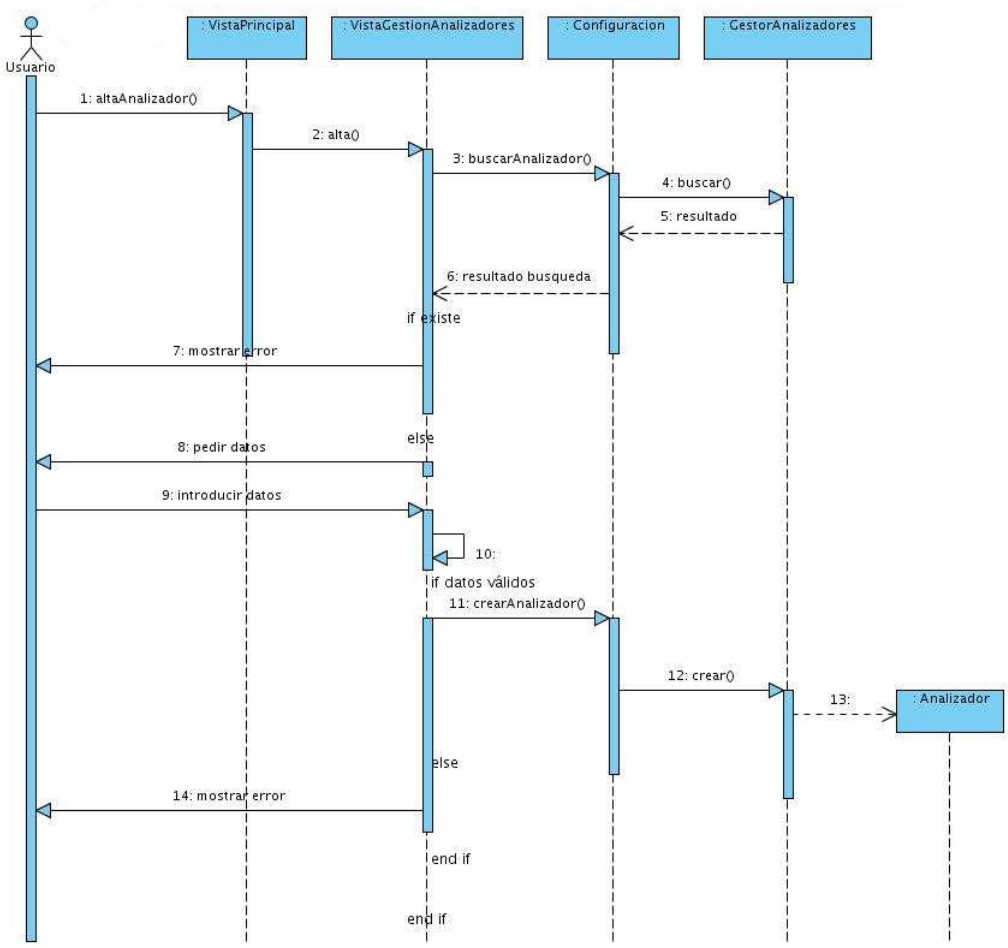

Figura 6.10: Diagrama de secuencia de diseño: Alta analizador

Gestión analizadores (Escenario 2): Baja analizador

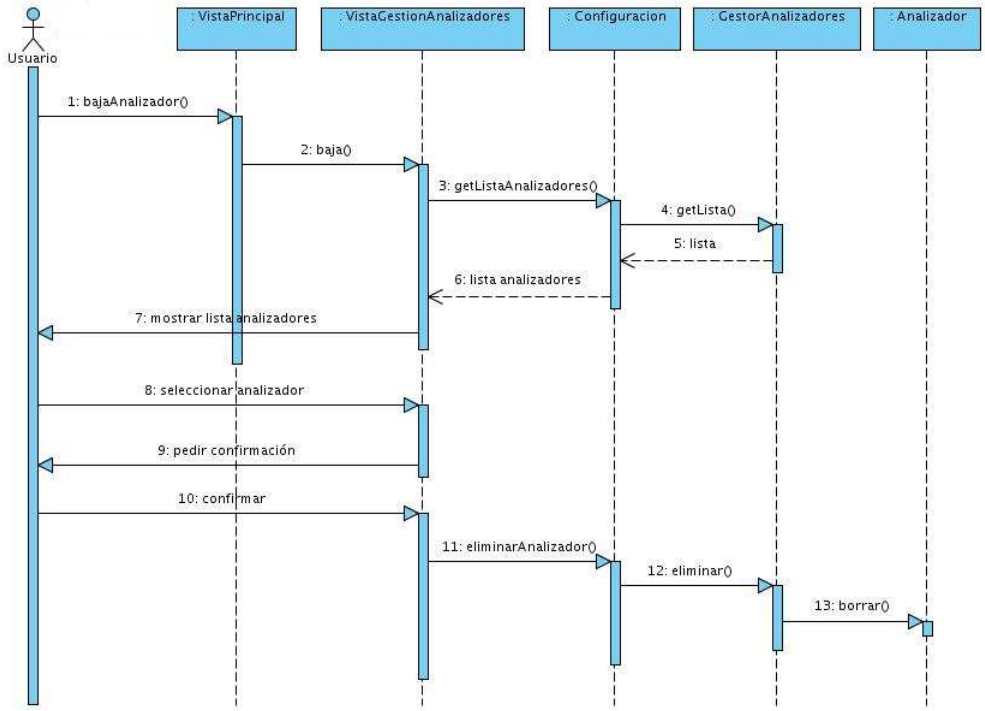

Figura 6.11: Diagrama de secuencia de diseño: Baja analizador

Gestión analizadores (Escenario 3): Consulta analizador

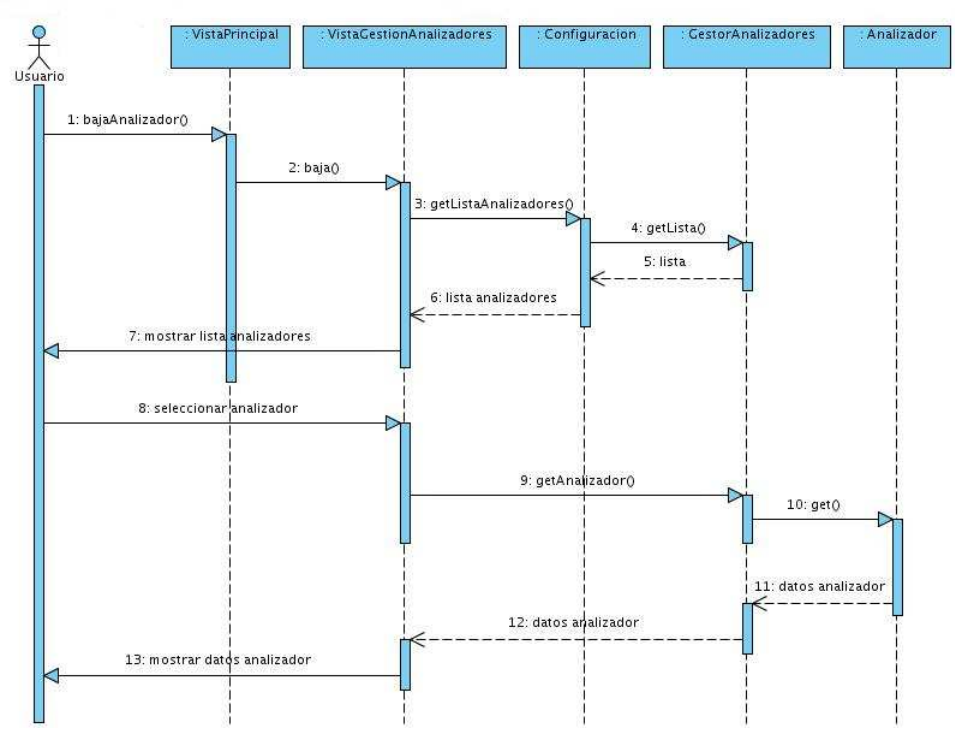

Figura 6.12: Diagrama de secuencia de diseño: Consulta analizador

Gestión analizadores (Escenario 4): Modificación analizador

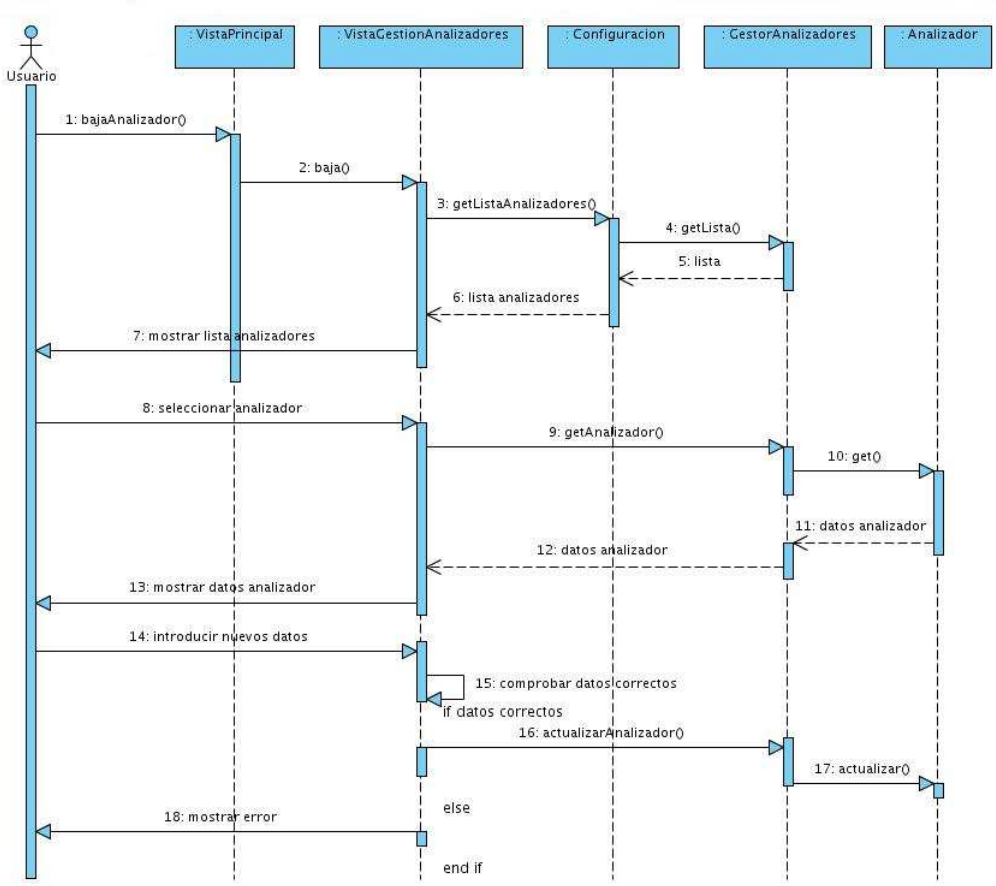

Figura 6.13: Diagrama de secuencia de diseño: Modificación analizador

Gestión analizadores (Escenario 5): Editar algoritmos

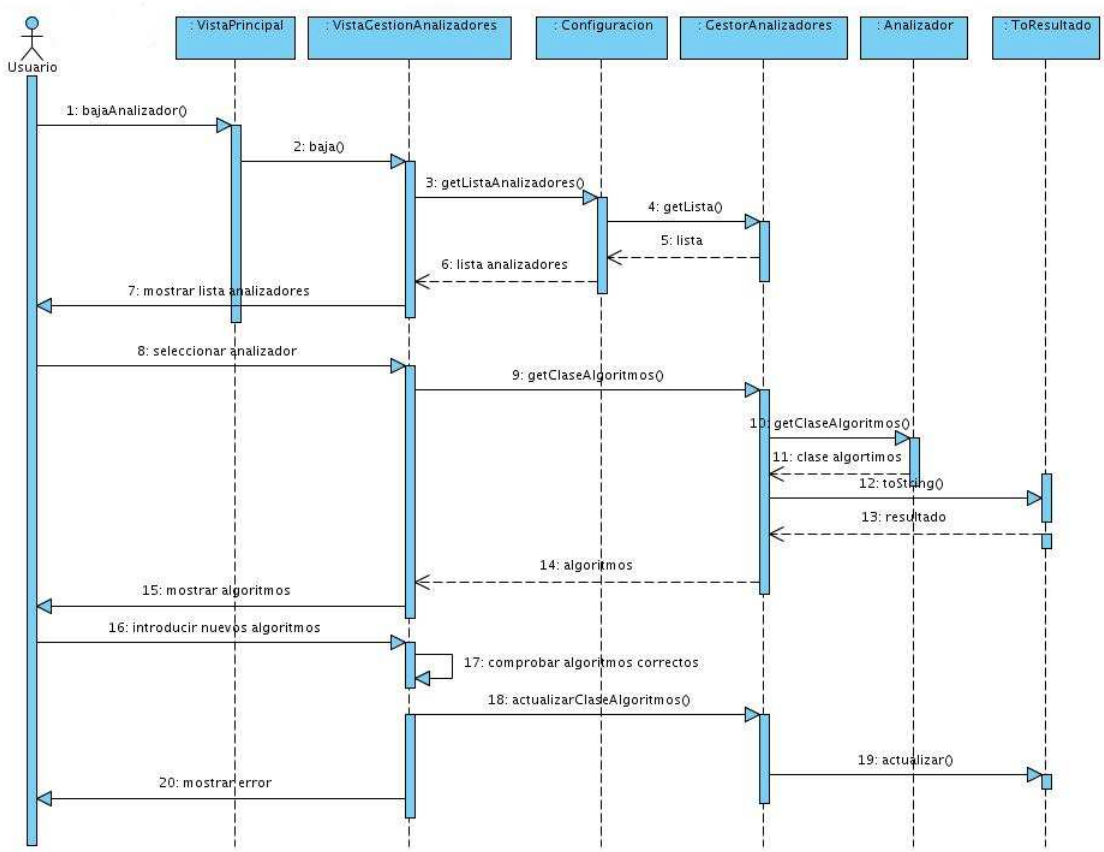

Figura 6.14: Diagrama de secuencia de diseño: Editar algoritmos

6.3. Patrones de diseño

Un patrón de diseño describe un problema que ocurre con frecuencia en ingeniería del *software* y detalla la solución a dicho problema, de manera que se pueda aplicar esta solución siempre que se produzca el problema.

La idea principal de los patrones es el concepto de estandarización de la información sobre un problema común y su solución.

Los patrones de diseño representan una evolución importante en la abstracción y reutilización del *software*. Estos dos conceptos son básicos en la programación. La abstracción representa la forma que tienen los desarrolladores para resolver problemas complejos dividiéndolos en otros más simples. La reutilización es igual de vital para el desarrollo de software.

Esta sección se citan y se describen brevemente los patrones de diseño que han servido de ayuda durante el desarrollo de este proyecto.

6.3.1. Patrón Singleton

El patrón *Singleton* es un patrón de creación cuyo objetivo es asegurar que una clase sólo tiene una instancia y proporcionar una forma de acceder a dicha instancia.

Para solucionar este problema, el patrón *Singleton* define un método de clase (estático) que permite acceder a la instancia única y que puede, al mismo tiempo, ser el responsable de crear la instancia.

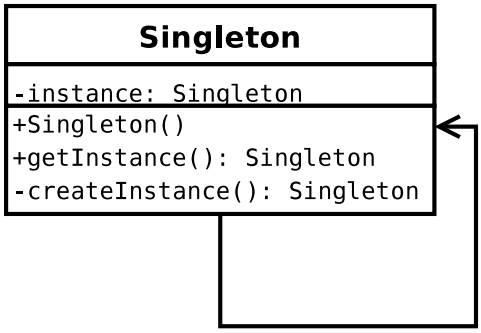

Figura 6.15: Patrón singleton

En el desarrollo de este proyecto, el patrón *Singleton* se ha empleado para la implementación de la clase *Configuración*, que maneja toda la información referente a la configuración del sistema. El uso de este patrón permite asegurar que sólo existe una instancia de la clase *Configuración* y facilita el acceso a la misma desde distintos puntos de la aplicación.

6.3.2. Patrón Estrategia

El patrón *Estrategia* es un patrón de comportamiento. Este tipo de patrones está relacionado con el flujo de control en un sistema. Ciertas formas de organizar el control en una aplicación pueden derivar en grandes beneficios para la eficiencia y el mantenimiento del sistema.

El propósito de este patrón es definir un grupo de clases que representan

un conjunto de posibles comportamientos. Estos comportamientos pueden ser fácilmente intercambiados en una aplicación, modificando la funcionalidad en cualquier instante.

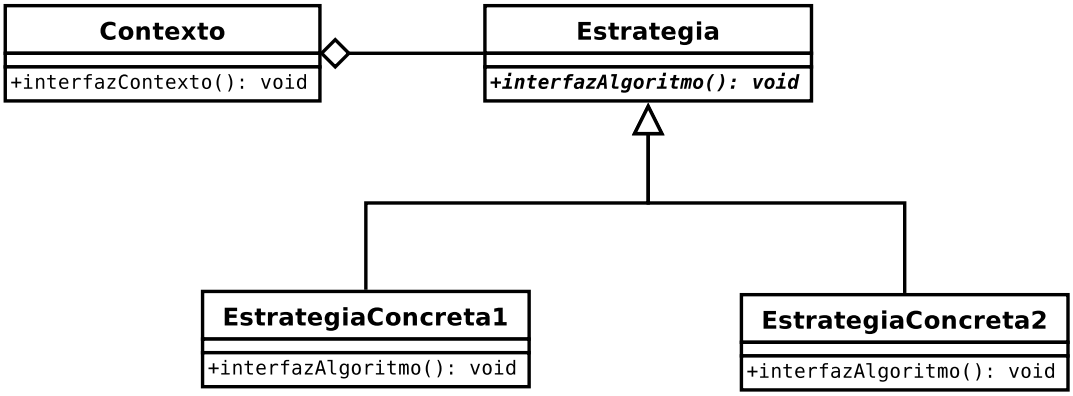

Figura 6.16: Patrón estrategia

El patrón *Estrategia* se ha usado en la implementación de los algoritmos de conversión al formato común para salida de análisis sintáctico y también de las llamadas a los distintos analizadores. En ambos casos, el uso de este patrón permite llevar a cabo las funcionalidades de los algoritmos, permitiendo al usuario abstraerse del analizador sintáctico concreto empleado.

6.3.3. Patrón Compuesto

El patrón *Compuesto* permite que los clientes traten de manera uniforme los objetos individuales y compuestos.

Para ello, compone los objetos en estructuras arbóreas, representando jerarquías de la forma parte-todo.

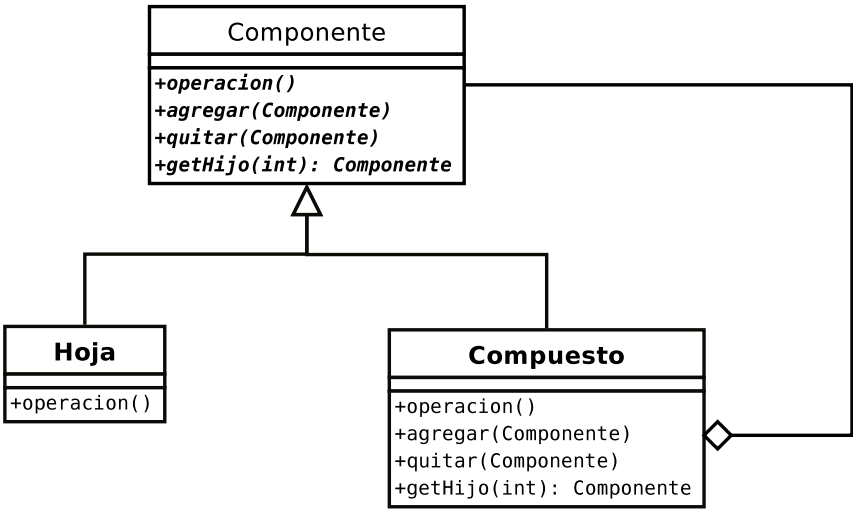

Figura 6.17: Patrón compuesto

El patrón *Compuesto* se utiliza para la implementación de los recursos sobre los

que se puede realizar un análisis sintáctico, es decir: frases y textos. El uso de este patrón permite tratar de la misma forma a una frase y a un texto (conjunto de frases) a la hora de realizar un análisis, haciendo mucho más sencilla la implementación.

Capítulo 7

Implementación y Pruebas

7.1. Implementación

7.1.1. Diagrama de componentes

El diagrama de componentes se utiliza para modelar la estructura del *software*, incluyendo las dependencias entre sus componentes, los componentes de código binario, y los ejecutables.

En el diagrama de componentes se modelan los correspondientes al sistema, en ocasiones agrupados por paquetes, y las dependencias que existen entre dichos componentes.

Aplicación

Este paquete, como su propio nombre indica, está formado por los componentes que dan lugar a la aplicación principal. Veamos brevemente en que consisten:

- Interfaz: Se refiere a todas las vistas que son presentadas al usuario y que permiten a éste comunicarse con la aplicación.
- Control: Es el componente que se encarga de llevar a cabo todo el control de negocio de la aplicación. Se comunica con la interfaz y también con los recursos y archivos manejados.
- ArhivoConfiguracion: Archivo XML que almacena toda la información clave para el funcionamiento de la aplicación. Ver 7.1.2.
- jgraph: Librería empleada para mostrar gráficamente los resultados de los análisis sintácticos.
- jdom: Librería empleada para manejar ficheros en formato XML.

Recursos

Este paquete representa a los recursos utilizados por el sistema para llevar a cabo los análisis sintácticos, es decir, los analizadores sintácticos y los archivos de configuración de los mismos.

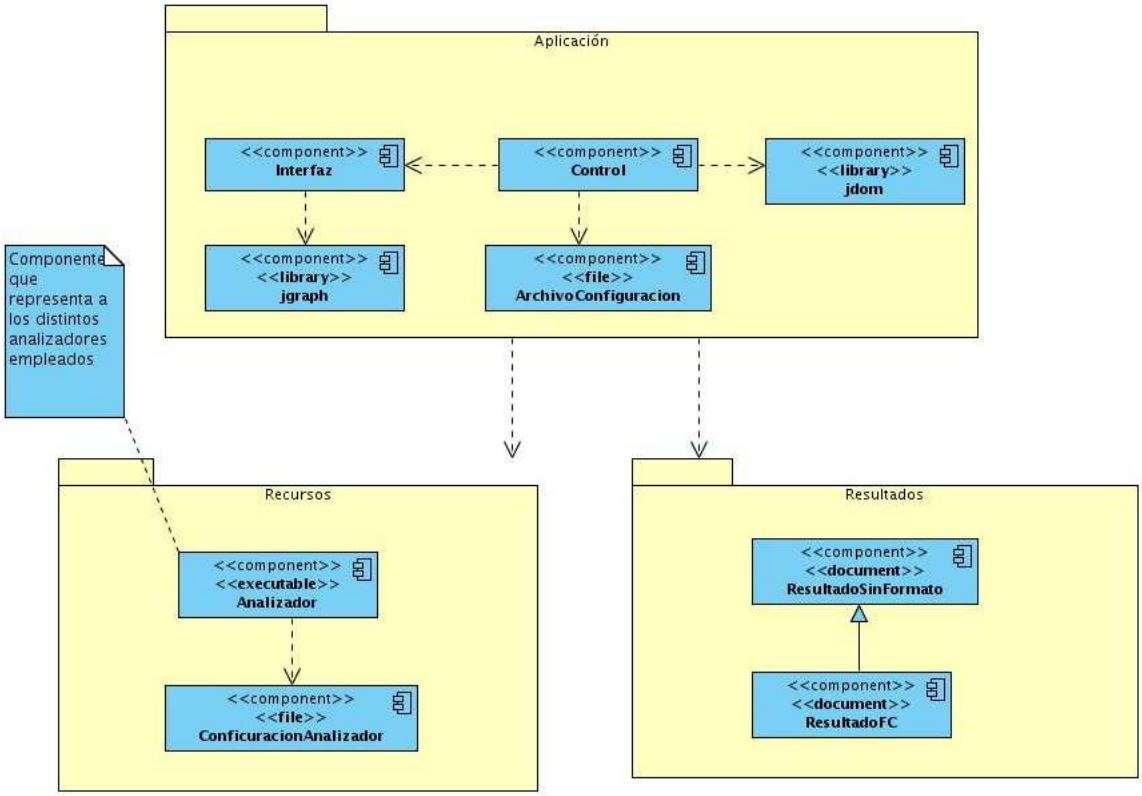

Figura 7.1: Diagrama de componentes

- Analizador: Ejecutable que representa a un analizador sintáctico utlizado por la aplicación.
- ConfiguracionAnalizador: Este componente representa al archivo de configuración propio de cada analizador. Ver 7.1.2.

Resultados

Este paquete contiene los componentes que representan los resultados del análisis, cuya obtención es el objetivo principal del sistema. Veamos estos componentes:

- ResultadoSinFormato: Componente que representa a los archivos que almacenan los resultados de los análisis sintácticos en un formato distinto del formato común propuesto.
- ResultadoFC: Archivos XML que almacenan el resultado de un análisis sintáctico en el formato común.

7.1.2. Interpretación de los archivos de configuración

Esta sección se centra en detallar los ficheros de configuración, parte esencial del sistema. La configuración de este sistema viene especificada, además de por el archivo de configuración general de la aplicación, por los archivos de configuración propios de cada analizador. A continuación se explican los contenidos de ambos.

Archivo de configuración del sistema

Este fichero contiene los datos fundamentales para el funcionamiento de la aplicación, como los directorios más importantes o la ubicación de los ficheros de configuración propios de cada analizador.Para una mejor comprensión del mismo se ha optado por describirlo a través del propio fichero XML y no a través de su DTD. El archivo de configuración tiene la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment/>
<entry key="directorio.base.default">
/home/adrian/NetBeansProjects/PFC_final/
</entry>
<entry key="directorio.salida.default">output/</entry>

<entry key="analizador.Alpage.nombre">Alpage</entry>
<entry key="analizador.Erial.nombre">Erial</entry>

<entry key="analizador.Erial.fichero">fichero_conf_Erial.xml</entry>
<entry key="analizador.Alpage.fichero">fichero_conf_Alpage.xml</entry>
</properties>
```

Como se puede observar, este documento XML sigue las pautas establecidas por el fichero *properties.dtd*. Esto es porque se usa la clase de *Properties*, incluída en el API de *Java* para manejar el fichero de configuración, facilitando enormemente el almacenamiento y recuperación de información de dicho archivo.

El elemento *directorio.base.default* contiene el directorio base por defecto, en el que se encuentra la aplicación. El elemento *dir.salida.default*, por el contrario, establece el directorio en el que, por defecto, se almacenan los resultados de los análisis.

Los elementos de la forma *analizador.nombre_analizador.nombre* especifican el nombre de un analizador, mientras que los elementos de la forma *analizador.nombre_analizador.fichero* especifican la ruta del fichero de configuración propio del analizador.

Archivos de configuración de los analizadores

Como ya se ha dicho, esta aplicación hace uso de ficheros de configuración para almacenar las características fundamentales de cada analizador, como la clase que contiene los algoritmos, los lenguajes disponibles o las correspondencias entre etiquetas del formato propio el analizador y las etiquetas del formato común. Veamos un ejemplo de este tipo de archivos:

```
<?xml version="1.0 "encoding="UTF-8"?>
<conf>
<class name="ErialToResultado">
<language name="Español"/>
<script name="script_erial.sh">
<param name=" "/>
</script>
</language>
<cat propio="DAMS" comun="det" />
<cat propio="V1SRI" comun="v" />
<cat propio="VRI" comun="v" />
<cat propio="NCMS" comun="nc" />
<cat propio="Sp00" comun="np" />
<cat propio="PY1S" comun="pro" />
<cat propio="AQMS" comun="adj" />
<cat propio="C" comun="conj" />
<cat propio="W" comun="adv" />
<cat propio="X" comun="prep" />
<cat propio="Fecha" comun="Fecha" />
<cat propio="Zg00" comun="Sigla" />
<cat propio="Abreviatura" comun="Abreviatura" />
<cat propio="Cifra" comun="Cifra" />
<cat propio="P" comun="P" />

<etiqueta propio="OD" comun="OD" />
<etiqueta propio="SUJA" comun="sujeto" />
<etiqueta propio="CNde" comun="CN" />
<etiqueta propio="CNen" comun="CN" />
<etiqueta propio="CNpara" comun="CN" />
<etiqueta propio="CNa" comun="CN" />
<etiqueta propio="CNpor" comun="CN" />
<etiqueta propio="CNante" comun="CN" />

</conf>
```

El archivo está compuesto por un elemento principal llamado *conf*, que contiene en su interior todos los parámetros necesarios para funcionamiento de un analizador sintáctico. En primer lugar, está el elemento *class*, que contiene el nombre de la clase que contiene los algoritmos encargados de lanzar el análisis sintáctico y de transformar la salida del mismo al formato común.

A continuación se pueden encontrar uno o varios elementos *languaje* (en función de si el analizador tiene soporte para uno o varios lenguajes). Este elemento contiene un atributo *name* que especifica el nombre del cada lenguaje. Este elemento, a su vez, contiene un elemento *script* que indica la ruta relativa del script que lanza la ejecución del analizador sintáctico.

En el archivo se pueden ecntrar una serie de elementos *cat*, que indican la correspondencia entre las categorías léxicas proporcionadas por la salida del analizador sintáctico y sus equivalentes en el formato común. Estos elementos tienen dos parámetros: *propio*, que repepresenta a una categoría léxica propia del analizador y *común*, su equivalente en el formato común.

Los elementos *etiqueta* representan la correspondencia entre etiquetas de dependencias entre palabras en el formato de salida del analizador y el formato común. Su funcionamiento es idéntico al de las categorías, y el significado de sus parámetros es el mismo.

7.2. Pruebas

En esta sección se detallan las pruebas realizadas para comprobar el correcto funcionamiento del sistema.

A lo largo del desarrollo de un proyecto de *software* pueden aparecer errores en los sitios más comunes. Por este motivo el proceso de pruebas es una parte fundamental de un desarrollo y a la cual el esfuerzo dedicado debe repartirse a lo largo de la implementación y en cada etapa del proceso.

Se puede definir una prueba como el proceso de ejecutar un programa con la intención específica de encontrar errores previos a la entrega al usuario final. El objetivo de las pruebas es encontrar defectos en el *software*

7.2.1. Pruebas de unidad

Estas pruebas se centran en cada unidad de software o módulo individualmente, realizando pruebas de caja blanca y de caja negra. Se tiene que probar que la información que fluye entre los módulos del sistema se hace de forma adecuada hacia y desde la unidad de programa que está siendo probada. Es necesario, por lo tanto, probar cada componente de forma individual.

Pruebas de caja blanca

Las pruebas de caja blanca se encargan de validar la lógica del módulo. Su realización debe garantizar que:

- Se recorren por lo menos todos los caminos independientes de cada módulo.
- Se ejecutan todas las decisiones lógicas en su parte verdadera y en su parte falsa.
- Se recorren todos sus bucles.

Todas estas pruebas fueron realizadas durante la implementación del sistema, comprobando que los módulos implementados cumplieran estos requisitos, y que realizan las operaciones para las cuales han sido codificados de forma correcta.

Pruebas de caja negra

Las pruebas de caja negra se centran en probar que se verifican los requisitos funcionales del software, es decir, que las funciones para las cuales se ha diseñado el sistema se cumplen sin errores. Los errores que se pretenden detectar mediante las técnicas de caja negra son:

- Funciones incorrectas o ausentes.
- Errores de rendimiento.
- Errores de inicialización o terminación.

Para verificar que todos estos errores no se producían en el sistema se probaron aspectos tales como:

Análisis sintáctico

Título:	**No seleccionar analizador**
Resultado:	Se muestra un mensaje de error, indicando que se debe seleccionar un analizador sintáctico como mínimo previamente
Evaluación	Correcto

Título:	**No introducir frases o textos para el análisis**
Resultado:	Se muestra un mensaje de error, indicando que se debe introducir una frase o texto para el análisis
Evaluación	Correcto

Título:	**Introducir un fichero incorrecto**
Resultado:	Se muestra un mensaje de error, indicando que el fichero introducido no es correcto
Evaluación	Correcto

Crear analizador

Título:	**Introducir un nombre ya existente para el analizdor**
Resultado:	Se muestra un mensaje de error, indicando que el nombre introducido ya existe
Evaluación	Correcto

Título:	**Introducir un nombre de archivo de configuración incorrecto**
Resultado:	Se muestra un mensaje de error, indicando que el archivo de configuración debe tener extensión .xml
Evaluación	Correcto

Título:	**Introducir un nombre incorrecto para la clase que contiene los algoritmos necesarios para la realización del análisis**
Resultado:	Se muestra un mensaje de error, indicando que el nombre de la clase debe seguir el formato válido para una clase *java* y que no debe existir previamente
Evaluación	Correcto

Título:	**No introducir ningún idioma**
Resultado:	Se muestra un mensaje de error, indicando que se debe introducir upor lo menos un idioma para el analizador
Evaluación	Correcto

Título:	**Introducir un script inexistente**
Resultado:	Se muestra un mensaje de error, indicando que el script introducido debe existir en el sistema
Evaluación	Correcto

Borrar analizador

Título:	**No seleccionar un analizador a eliminar**
Resultado:	Se muestra un mensaje de error, indicando que se debe seleccionar el analizador a borrar
Evaluación	Correcto

Título:	**Se selecciona un analizador pero se decide cancelar la operación**
Resultado:	Se vuelve a la vista de *Gestión de analizadores*
Evaluación	Correcto

Modificar analizador

Título:	**No seleccionar un analizador a modificar**
Resultado:	Se muestra un mensaje de error, indicando que se debe seleccionar el analizador a modificar
Evaluación	Correcto

Título:	**No se introduce ningún idioma**
Resultado:	Se muestra un mensaje de error, indicando que se debe introducir por lo menos un idioma para el analizador
Evaluación	Correcto

Título:	**Introducir un script inexistente**
Resultado:	Se muestra un mensaje de error, indicando que el script introducido debe existir en el sistema
Evaluación	Correcto

Editar algoritmos

Título:	**Se introducen errores en el código (en la clase implementada por el usuario)**
Resultado:	Se muestra un mensaje de error, indicando donde se han producido los errores.
Evaluación	Correcto

7.2.2. Pruebas de integración

El principal objetivo de este tipo de pruebas es detectar fallos en la comunicación e interacción de las clases del sistema. Se incluye aquí además la integración tanto de diversas tecnologías en la aplicación como la integración de herramientas externas a las clases Java propiamente dichas.

De este modo tomando los módulos probados en las pruebas de unidad se trata de ver si en la estructura construida a partir de las clases, fluye la información de forma correcta y esperada.

Durante la implementación las clases fueron probadas de manera individual y progresivamente en su conjunto. Con la aplicación ya acabada se procedió a un conjunto de pruebas globales obteniendo unos resultados satisfactorios.

7.2.3. Pruebas de validación

Se han realizado múltiples pruebas para asegurar que los documentos XML existentes en el sistema sean válidos. La validación de estos documentos se centra en su estructura. Se ha comprobado que los documentos XML estén estructurados de acuerdo al DTD asociado.

Parte III

Manual de usuario

Capítulo 8

Instalación de la aplicación

En esta sección se describe el proceso de instalación de la aplicación, así como los recursos *software* y *hardware* necesarios para el funcionamiento de la misma.

8.1. Recursos necesarios

8.1.1. Recursos hardware

A continuación se describen brevemente los recursos *hardware* necesarios para el funcionamiento de la aplicación.

- CPU: 400 Mhz o superior (recomendable 1.5GHz). El tipo de procesador es indiferente, el único requisito es que exista una máquina virtual de *Java* para dicho procesador.
- Memoria: 512MB (recomendable 1GHz).
- Disco duro: 20 GB.

8.1.2. Recursos software

En este apartado se describen los distintos recursos *software* necesarios para el correcto funcionamiento de la aplicación.

- Sistema operativo: Si bien la aplicación está implementada en *java*, que es multiplataforma, es obligatorio el uso de un sistema operativo basado en GNU/LINUX, ya que la mayoría de las herramientas empleadas para el análisis sintáctico están desarrolladas exclusivamente para este tipo de plataformas. Concretamente se recomienda el uso de una distribución basada en *Debian* como, por ejemplo, *Ubuntu*, usada durante el desarrollo de la aplicación.
- Máquina virtual Java: Para el correcto funcionamiento de la aplicación es necesario el uso de la *Máquina Virtual Java* (JVM) en su versión 1.6 o superiores.

- Librerías: Son necesarias distintas librerías o API's *Java* para que la aplicación pueda funcionar:, como JDOM para manejar ficheros xml o JGRAPH para realizar los gráficos empleados para mostrar los resultados de los análisis sintácticos.

- Analizadores sintácticos: La aplicación necesita que haya instalados en el equipo herramientas de análisis sintáctico para poder llevar a cabo el trabajo para el que ha sido desarrollada. Se han probado diversos analizadores, como *Erial* (ver 10) o *Alpage* (ver 11), pero el usuario puede añadir otros nuevos.

8.2. Instalación de los programas necesarios

8.2.1. Instalación de la máquina virtual Java

En primer lugar es necesario descargar el jdk del sitio web de SUN (http://java.sun.com/j2se). Si se trabaja bajo Debian hay que bajarse el fichero con extensión .bin.

El proceso de instalación accederá a ciertos directorios para los cuales es necesario tener permisos de administrador, por lo tanto se abre una consola y se introduce:

```
sudo -s
```

. A continuación se solicita la contraseña, habrá que introducir la del administrador.

Se dan permisos de ejecución al archivo que se acaba de descargar:(nota: en la notación se pone <version> el usuario hará coincidir ese valor con la versión que se descargue):

```
jdk-<version>-linux-i586.bin
```

A continuación se copia dicho archivo a /usr/local/jdk (si no existe hay que crear este directorio) y se ejecuta, mediante el comando:

```
./jdk-<version>-linux-i586.bin
```

Una vez instalado es necesario actualizar la variable PATH del sistema añadiéndole la ruta del directorio donde se acaba de instalar el jdk. Para ello, el usuario puede acceder al .bashrc de su home o bien editar el fichero */etc/profile* (para lo cual necesitará permisos de administrador).

8.2.2. Instalación del analizador sintáctico Erial

Erial es uno de los analizadores sintácticos utilizados por la aplicación. Para detalles sobre su instalación y uso consultar la sección 10.

8.2.3. Instalación del analizador sintáctico Alpage

Alpage es uno de los analizadores sintácticos utilizados por la aplicación. Para detalles sobre su instalación y uso consultar la sección 11.

8.2.4. Instalación de la aplicación

Para poder utilizar la aplicación *Representación de Analizadores en Plataformas Transversales de prOcesamiento y Recuperación de información* (RAPTOR) basta con compiar el directorio RAPTOR del cd adjunto.
Una vez hecho esto, para ejecutar la aplicación tendremos que ejecutar el script *raptor.sh* presente en dicho directorio.

Capítulo 9

Uso de la aplicación

En este capítulo se describe de la manera más detallada posible la forma correcta de uso de la aplicación. Las explicaciones irán acompañadas de ilustraciones para facilitar la comprensión de las mismas.

9.1. Lanzamiento de la aplicación

Este es el aspecto que presenta la aplicación una vez lanzada su ejecución:

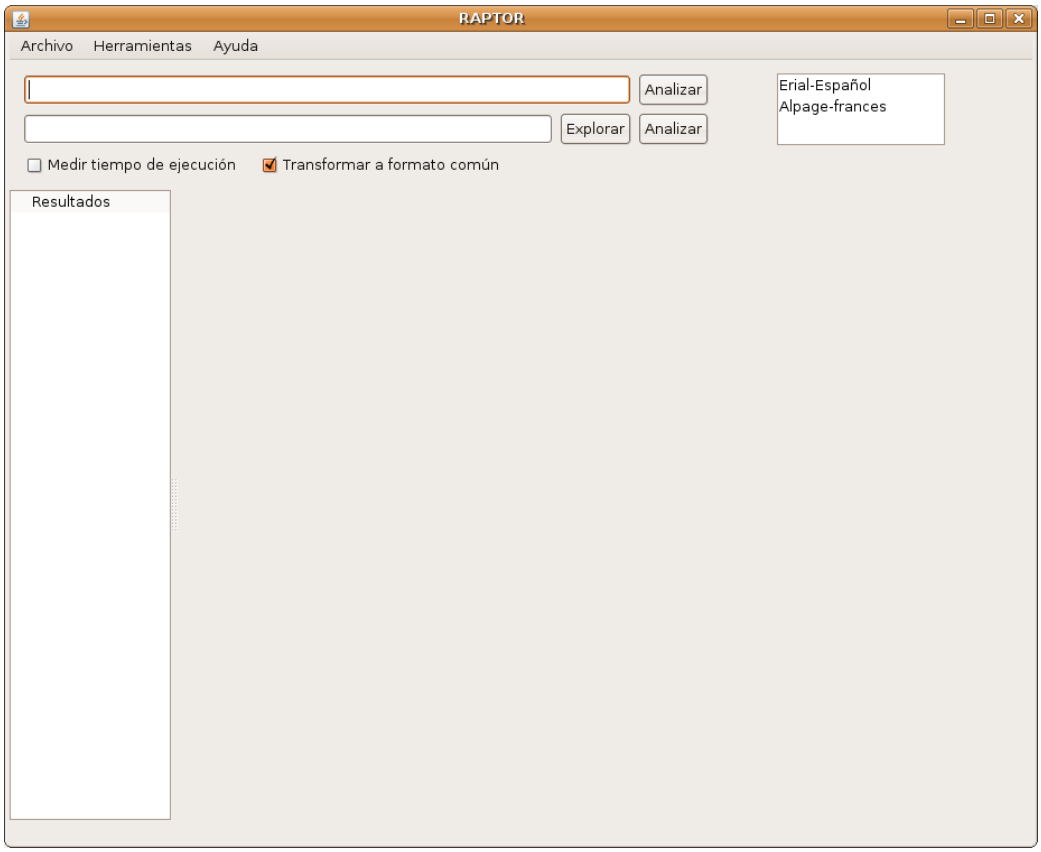

Figura 9.1: Ventana Principal de la aplicación

En la parte superior se puede observar una *barra de menú*, que permite realizar distintas acciones. Contiene los siguientes menús:

- Archivo: Contiene la opción *Salir* que, como su porpio nombre indica nos permite abandonar la aplicación.
- Herramientas: Contiene las siguientes opciones: *Preferencias*, que nos permite editar las opciones de configuración del sistema, y *Gestionar analizadores*, que sirve para poder llevar una completa gestión de los analizadores sintácticos.
- Ayuda: Contiene la opción *Acerca de*, que nos muestra información acerca de la aplicación *Raptor*.

9.2. Análisis sintáctico

El objetivo principal de la aplicación es poder llevar a cabo análisis sintácticos de textos o frases empleando distintos analizadores y mostrar los resultados, tanto en un formato común como en el formato propio de cada analizador.

En esta sección se explica cómo se pueden llevar a cabo tales acciones y cómo se pueden consultar los resultados obtenidos.

9.2.1. Lanzar el análisis sintáctico

En la siguiente imagen se muestra la vista de análisis. Ésta es mostrada al iniciar la aplicación, pero también se puede acceder a ella pulsando el botón de análisis, situado en la esquina superior izquierda, debajo de la barra de menú.

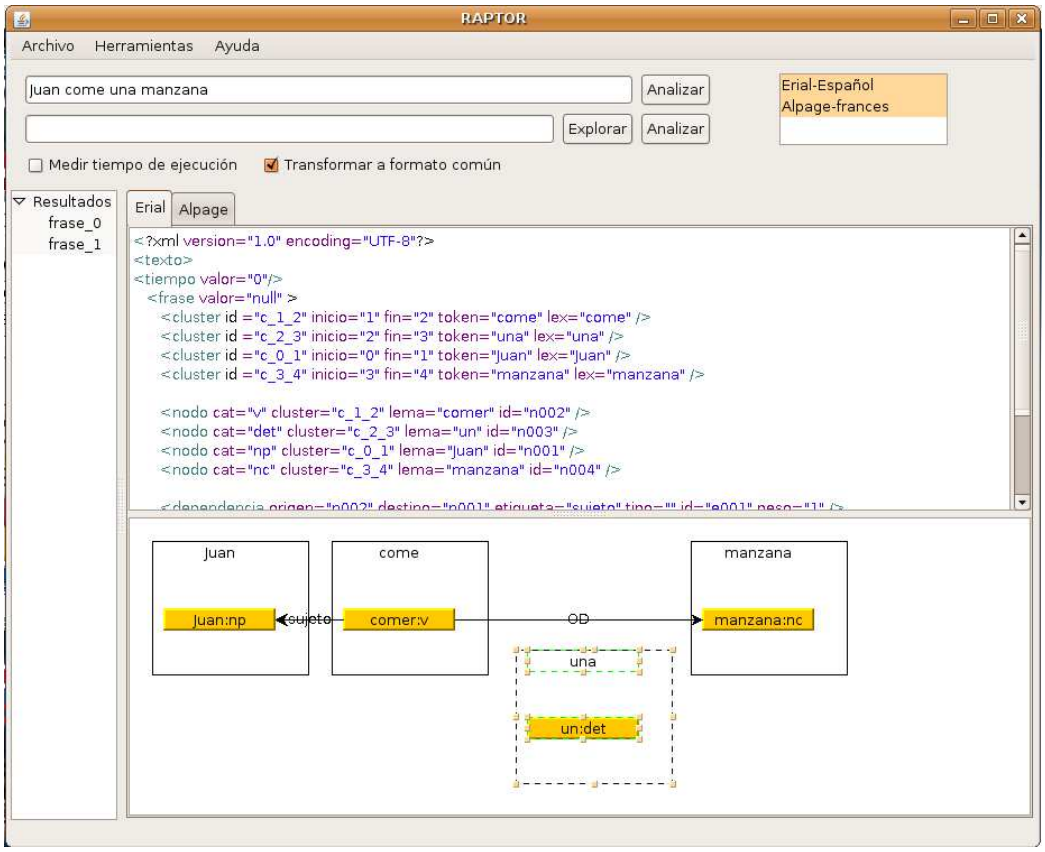

Figura 9.2: Vista de análisis sintáctico

Lo primero que nos llama la atención es el formulario que sirve para introducir los datos necesarios para el análisis, que se puede observar en la siguiente figura:

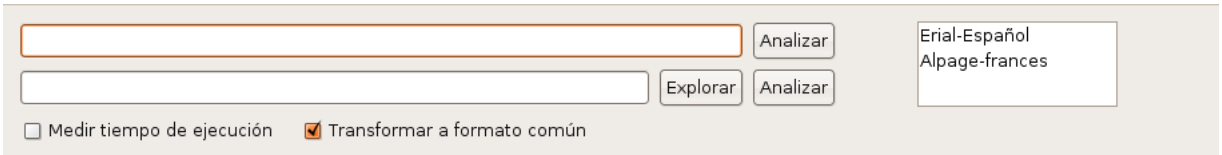

Figura 9.3: Formulario de análisis sintáctico

Contiene dos campos de texto: El superior, sirve para introducir directamente la frase que queremos analizar, mientras que el inferior permite introducir la ruta de un fichero de texto que contiene un conjunto de frases a analizar. Pulsando el botón *Explorar* podemos seleccionar el fichero que queramos en el sistema de archivos.

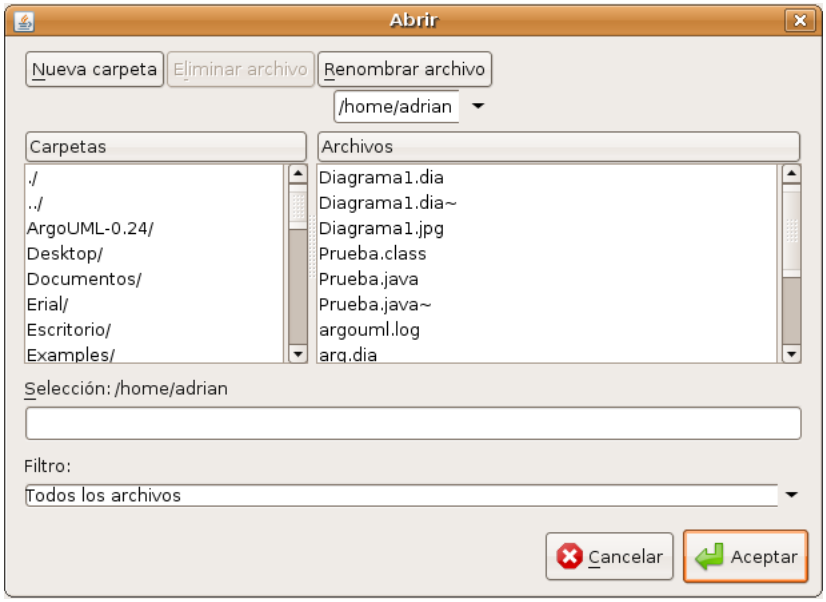

Figura 9.4: Diálogo de selección de archivo de texto para el análisis

Sin embargo, antes de realizar un análisis sintáctico se debe seleccionar el analizador o los analizadores que queramos emplear para tal fin. Si no hacemos esto, se nos mostrará el siguiente mensaje de error:

Figura 9.5: Mensaje de error

Esto se hace mediante la lista seleccionable de la derecha. Si queremos seleccionar más de un analizador sintáctico simultáneamente debemos mantener pulsada la tecla *Control* mientras seleccionamos los distintos analizadores.

Figura 9.6: Lista seleccionable de analizadores

Por último, podemos encontrar en la parte inferior dos casillas de verificación. La primera nos permite especificar si queremos medir el tiempo empleado en el análisis sintáctico o no, mientras que la segunda sirve para indicar de que manera queremos que se nos presenten los resultados: en el formato propio del analizador o en el formato común.

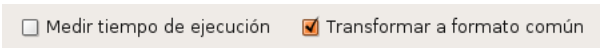

Figura 9.7: Casillas de selección de opciones de análisis

9.2.2. Interpretar los resultados del análisis sintáctico

Una vez realizado el análisis sintáctico, llega la hora de consultar los resultados obtenidos. La siguiente imagen muestra un ejemplo de resultados de un análisis sintáctico.

En la parte derecha se puede observar un árbol que contiene todas las frases

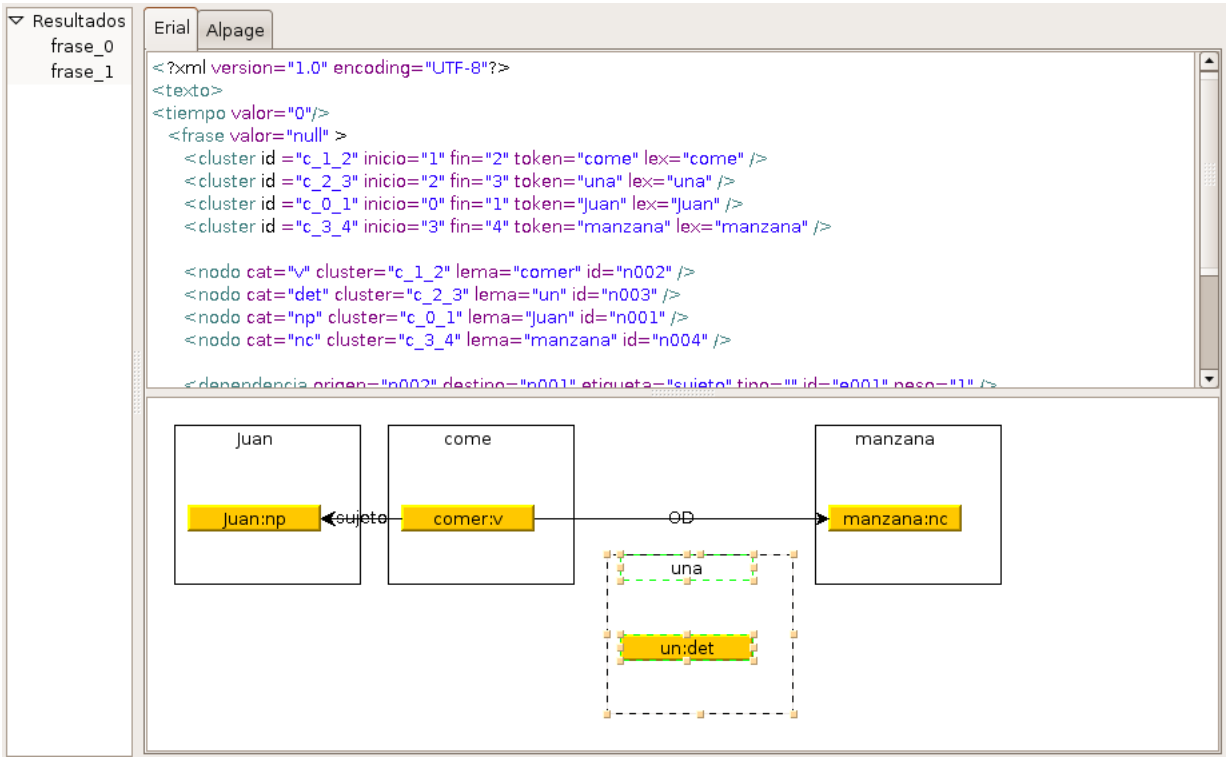

Figura 9.8: Vista de resultados del análisis

analizadas. Pulsando en una frase determinada, se mostrará el resultado del análisis de dicha frase.

En la parte central se muestra el resultado del análisis sintáctico de una frase. En el caso de que se hayan lanzado varios analizadores, pulsando las pestañas de la parte superior podemos intercambiar entre los resultados obtenidos por cada analizador.

En el caso de que se haya optado por mostrar los resultados en el formato propio del analizador, se mostrará simplemente ese resultado en formato de texto, tal como se puede observar en la siguiente imagen:

En caso de que se muestren los resultados en el formato común, el panel

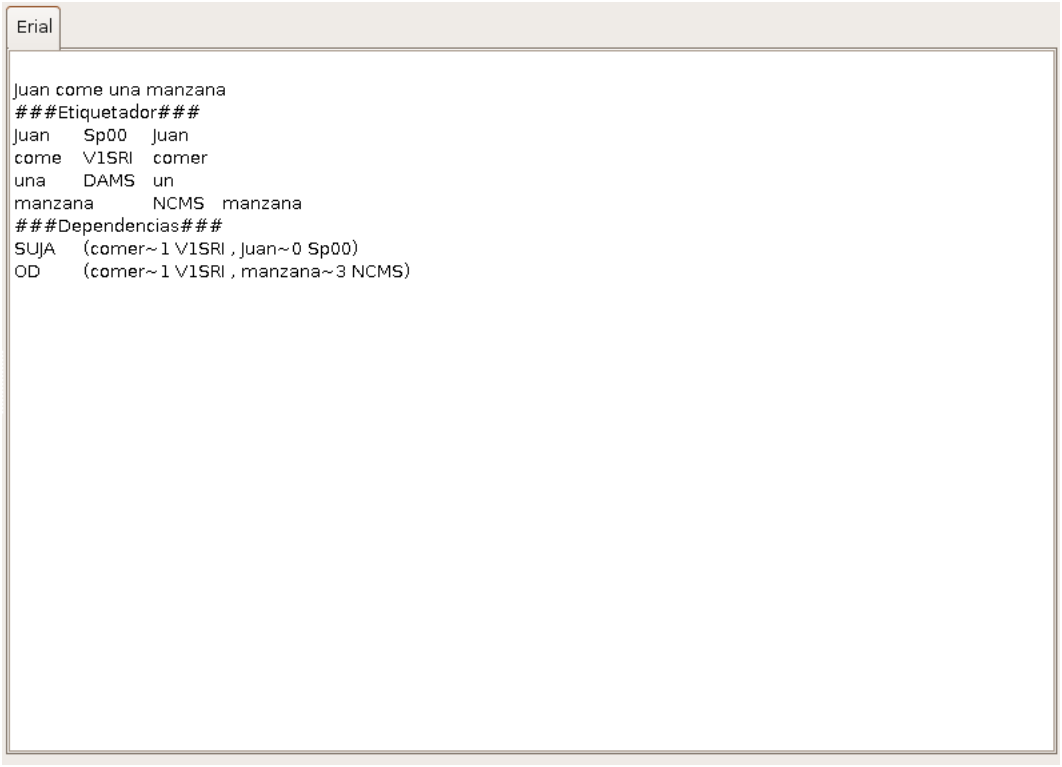

Figura 9.9: Vista de resultado sin convertir al formato común

será dividido en dos partes:

- En la parte superior se mostrará el archivo xml resultante.
- En la parte inferior se mostrará un gráfico que sigue el formato descrito en el apartado 1.5.1. Los componentes del gráfico podrán moverse al antojo del usuario para facilitar su comprensión.

9.3. Gestión de la configuración

Para poder consultar y modificar las opciones de configuración de la aplicación se debe seleccionar la opción *Preferencias* del menú *Herramientas*, en la barra de menús. Esto provocará que se nos muestre la siguiente ventana:

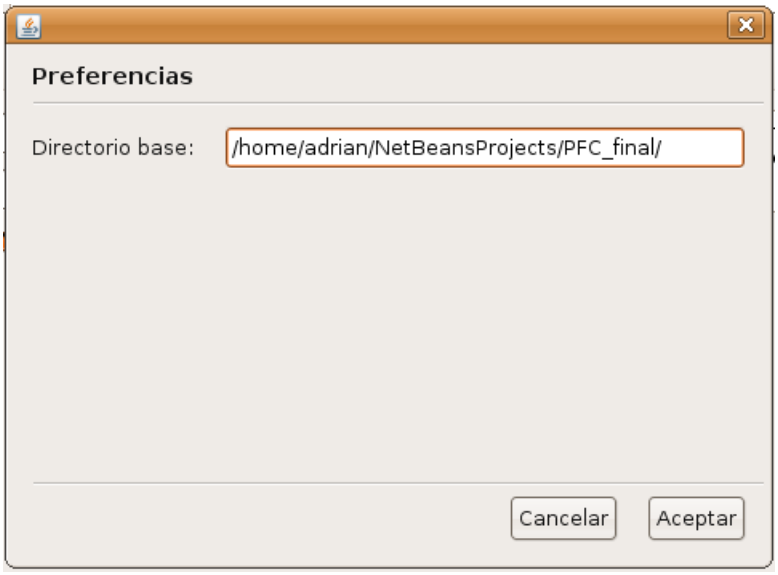

Figura 9.10: Edición de la Configuración

En ella se muestran los parámetros principales de la aplicación, cuya modificación no es muy recomendada. Sin embargo, si el usuario lo cree necesario, puede introducir los valores que crea conveniente y pusar el botón *Aceptar*, almacenando los cambios realizados.

9.4. Gestión de analizadores sintácticos

Para poder gestionar los distintos analizadores sintácticos a utilizar se debe seleccionar la opción *Gestionar Analizadores* del menú *Herramientas*. Una vez hecho esto se abrirá la siguiente ventana:

Figura 9.11: Ventana de gestión de analizadores

En la parte izquierda se puede observar una estructura en forma de árbol que nos permite seleccionar un analizador existente para realizar cualquier operación de gestión sobre él.

En la parte central de la ventana se nos muestra información sobre el analizador seleccionado, como su nombre, su fichero de configuración o la clase de sus algoritmos.

En la parte inferior de la ventana, podemos encontrar un conjunto de botones que permiten realizar todas las operaciones de gestión disponibles, que se explicarán a continuación: Crear un nuevo analizador o modificar, borrar o editar los algoritmos de un analizador sintáctico ya existente.

9.4.1. Crear un nuevo analizador

Una vez pulsado el botón correspondiente, mencionado en el apartado anterior, se mostrará la ventana de creación de un nuevo analizador. A continuación se detallará el proceso a llevar a cabo. Se usarán los botones *Siguiente* y *Anterior* para ir al paso posterior o previo, respectivamente, el botón *Cancelar* para abortar la operación, el botón *Terminar* para finalizar y el botón *Restablecer* para recuperar los valores iniciales.

Paso 1

En primer lugar, lo que se debe hacer es seleccionar un nombre para el nuevo analizador. En caso de que el nombre seleccionado ya exista o no sea correcto se mostrará un mensaje de error.

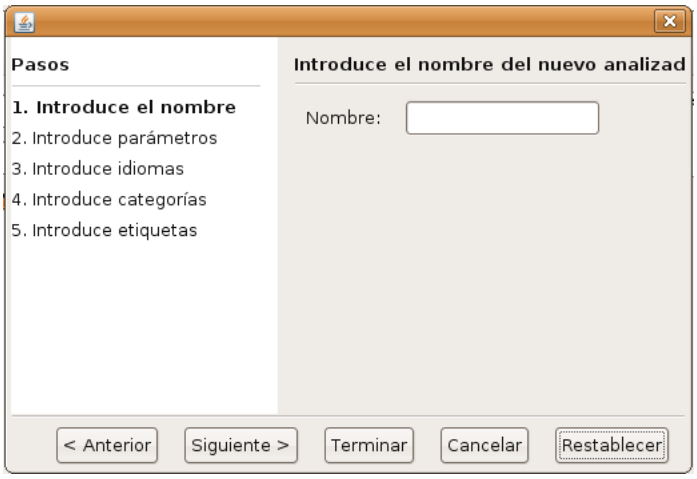

Figura 9.12: Crear analizador: paso 1

Paso 2

Una vez introducido el nombre, se nos pedirá que introduzcamos varios parámetros:

- Fichero: Ruta donde queremos que se almacene el fichero de configuración del analizador. Debe tratarse de un fichero XML.

- Clase: Nombre que le queremos dar a la clase que contiene los algoritmos necesarios para el funcionamiento del analizador. Debe ser un nombre válido para una clase *Java* y no existir previamente.

- Tipo: Tipo de analizador, puede ser *Java* o de otro tipo.

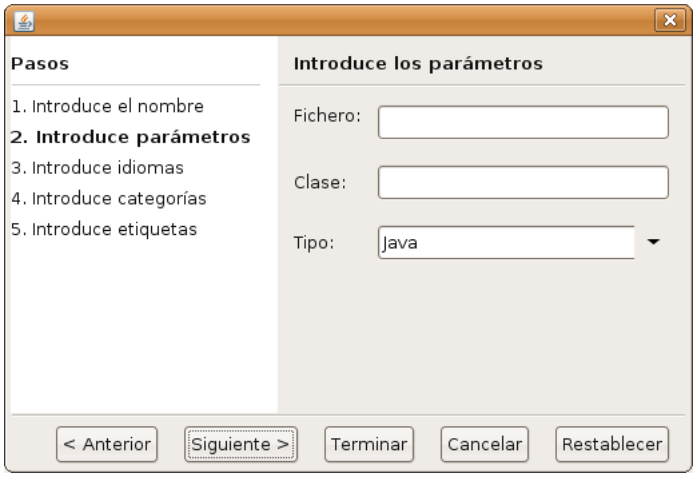

Figura 9.13: Crear analizador: paso 2

Paso 3

A continuación se deben introducir los distintos idiomas que soporta el analizador (primer campo de texto), asociando cada uno con el script correspondiente que ejecuta el analizador para dicho idioma(segundo campo de texto). Se puede usar el botón *Explorar* para seleccionar dicho script en el sistema de archivos. Para añadir un par idioma-script se debe pulsar el botón *Añadir*. Para modificar o borrar un par ya añadido, se debe seleccionar éste en la lista y, a continuación, pulsar el botón correspondiente. Los botones *Subir* y *Bajar* sirven para alterar el orden de la lista, mientras que el botón *Restablecer* sirve para vaciar la lista.

Figura 9.14: Crear analizador: paso 3

Paso 4

En cuarto lugar, se deben introducir las correspondencias entre las categorías léxicas del analizador y su equivalente en el formato común. El funcionamiento es prácticamente idéntico al de la lista de idiomas, siendo el primer campo de texto una categoría propia del analizador y el segundo su equivalente en el formato común.

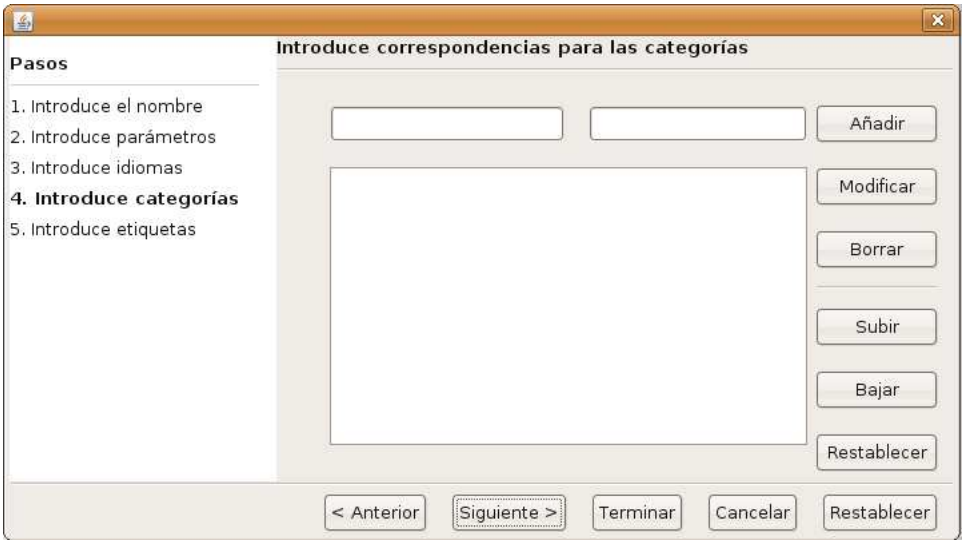

Figura 9.15: Crear analizador: paso 4

Paso 5

Por último, se deben introducir las correspondencias entre las etiquetas para las dependencias del analizador y su equivalente en el formato común. El funcionamiento es idéntico al caso anterior.

Después de introducir todas las etiquetas, se debe pulsar el botón *Terminar* para finalizar este proceso y dar de alta el nuevo analizador en el sistema.

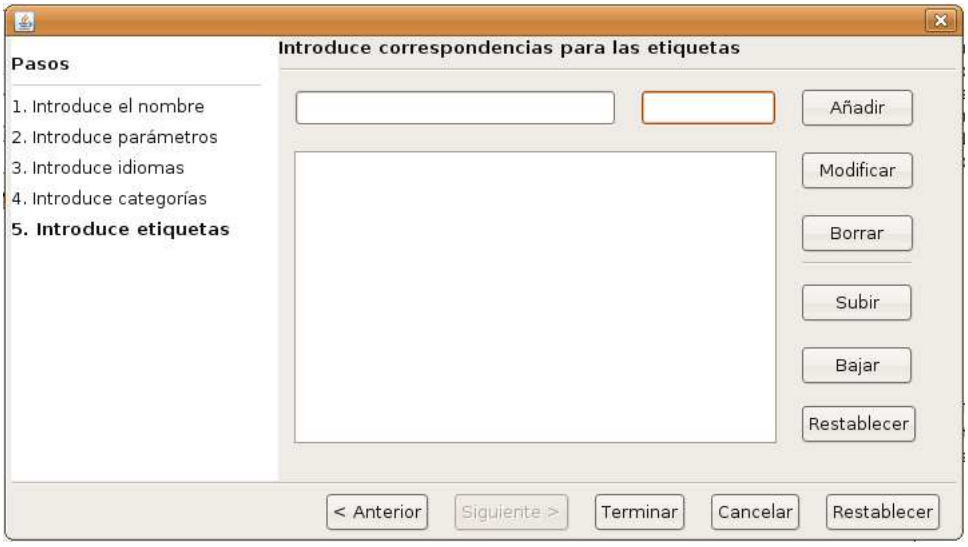

Figura 9.16: Crear analizador: paso 5

9.4.2. Modificar un analizador

Para modificar un analizador previamente se ha de selecionar en el árbol de la izquierda. Una vez hecho esto, se pulsa el botón *Modificar* Dando lugar a la siguiente ventana:

El proceso es muy similar al del alta, con dos diferencias fundamentales:

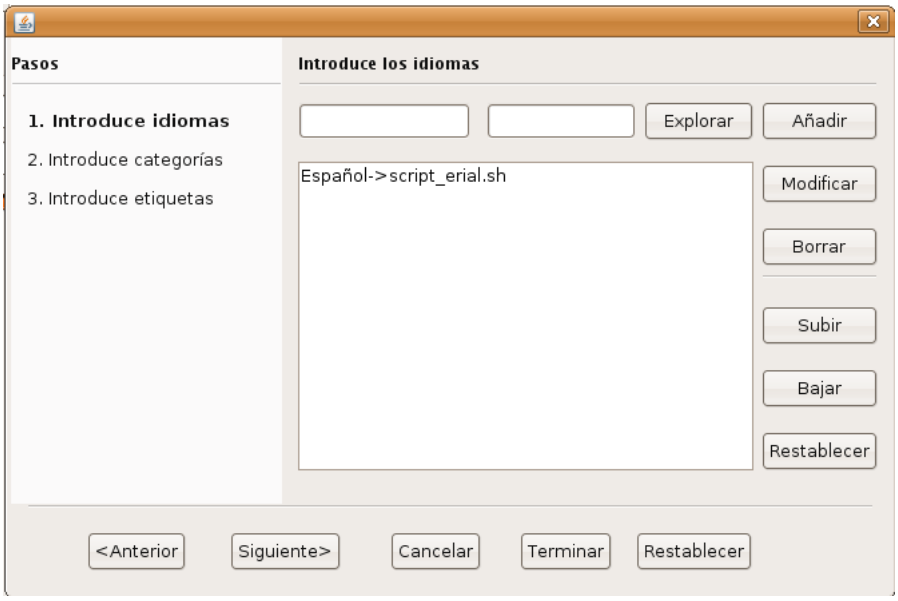

Figura 9.17: Modificar analizador

1. Los dos primeros pasos son suprimidos, ya que poder modificar el nombre o la ruta del fichero de configuración traería más problemas que ventajas.

2. El formulario contiene inicialmente los valores actuales del analizador. Si se

pulsa el botón *Restablecer* se recuperarán esos valores, y no se vaciarán los campos como en el caso del alta.

9.4.3. Borrar un analizador

Para eliminar un analizador previamente se ha de selecionar en el árbol de la izquierda. Una vez hecho esto, se pulsa el botón *Borrar*, dando lugar a la siguiente ventana, que nos pide confirmación para borrar dicho analizador.

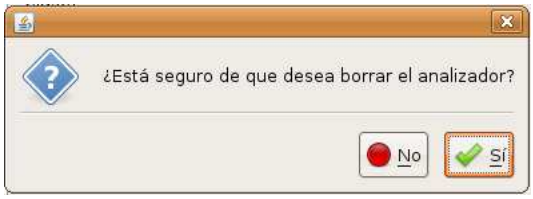

Figura 9.18: Borrar analizador

9.4.4. Editar algoritmos de un analizador

Para editar los algoritmos de conversión al formato común y de lanzamiento del análisis, previamente se ha de selecionar el analizador en el árbol de la izquierda. A continuación se debe pulsar el botón *Editar algoritmos* lo que provocará que se habra una nueva ventana con un editor java:

En la parte superior podemos observar dos botones:

- Guardar: Tiene forma de disquete y sirve para almacenar los cambios realizados.
- Compilar: Está representado por unas herramientas y sirve para lanzar el compilador.

El editor tiene dos componentes fundamentales. El primero, situado en la parte superior, es el editor propiamente dicho, donde se puede modificar el código *Java* para modificar los algoritmos.

Debajo del *editor* se halla el *panel de salida*, que muestra el resultado de las compilaciones realizadas, indicando si son o no correctas y, en el caso de que no lo sean, mostrando los errores encontrados.

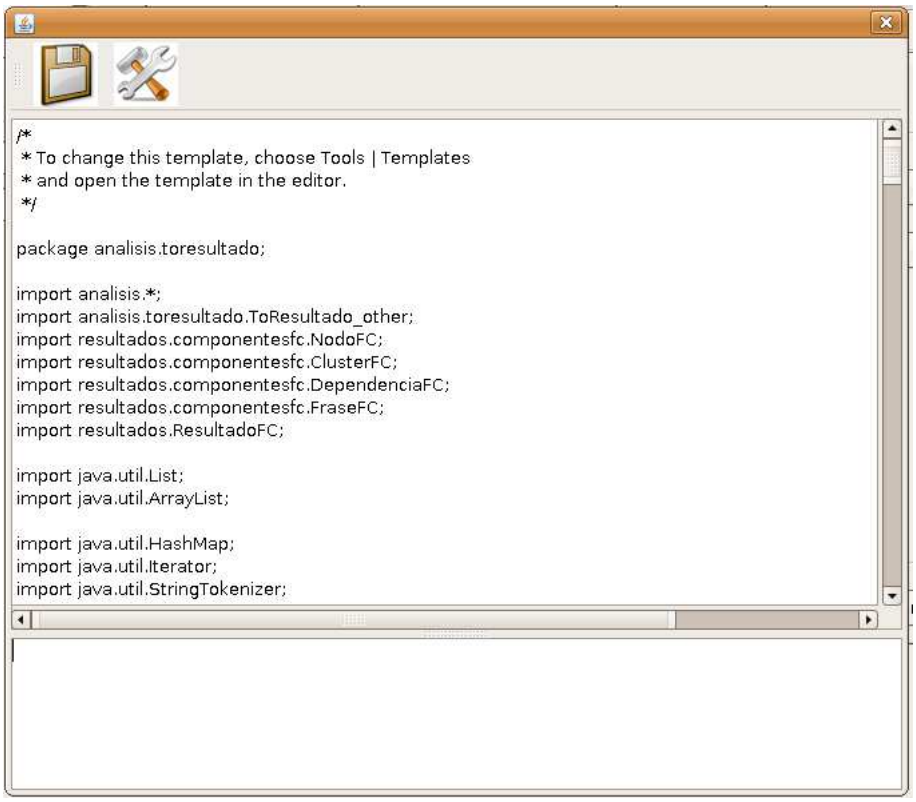

Figura 9.19: Editor java

Parte IV

Apéndices

Capítulo 10

Apéndice A: Erial

10.1. Introducción

Erial es un entorno para la recuperación de información basado en técnicas de PLN y desarrollado en el marco del proyecto *Erial*[13] (Extracción y Recuperación de Información mediante Análisis Lingüístico), un proyecto de investigación que aspira a mejorar los resultados de las aplicaciones de RI en gallego y español mediante el uso de técnicas de PLN. Ha sido desarrollado por un grupo de investigación de la Universidad de A Coruña que aportaba el conocimiento computacional, las Universidades de Santiago de Compostela y de Vigo y el centro Ramón Piñeiro para la investigación en Humanidades, que aportaban el conocimiento lingüístico.

10.2. Funcionamiento

El recurso Erial es un analizador sintáctico superficial basado en una cascada de *autómatas finitos* [7]. En este tipo de análisis sólo se devuelve un camino posible, es decir, no se tienen en cuenta ambigüedades. La forma de representar el contenido de los sintagmas se hace por medio de dependencias de las palabras que conforman los mismos. Concretamente reduce el número de palabras en una dependencia a dos, de ahí que el trabajo se realice sobre pares de dependencias. Estas dependencias incluyen los lemas de las palabras, esta normalización es muy útil puesto que establece una base común para las formas semánticamente equivalentes en las frases.

Concretamente la arquitectura seguida se basa en el uso de cinco capas, cuya entrada por defecto es el etiquetador-lematizador *MrTagoo*[8] previamente preprocesado. Como se describirá más adelante en el caso del proyecto la entrada fue proporcionada por el etiquetador-lematizador *TreeTagger*, previo preprocesamiento, y después de aplicarle un conjunto de cambios para su adaptación a la entrada esperada por el analizador sintáctico.

A continuación se describe brevemente el cometido de cada etapa, pero antes es conveniente destacar que estamos tratando con reglas, cuya parte derecha deberá coincidir con un patrón para poder ser sustituida por su parte izquierda. Por ejemplo para un sintagma nominal sencillo:

Sintagma Nominal − >Determinantes Sustantivo (SN − > Det N)

El analizador deberá encontrar en su entrada un conjunto de *Det* y *N* para poder sustituir este par por SN.

A continuación se pasa a enumerar las distintas capas de la arquitectura:

1. La primera capa mejora la salida del *pre-procesador* (10.3) eliminando el ruido generado por ciertas construcciones sintácticas. Se puede considerar como una capa de pre-procesamiento antes de iniciar el análisis propiamente dicho.

2. Tratamiento de cifras que están total o parcialmente en formato no numérico. Por ejemplo, *"cinco millones"*o *"5 millones"*.

3. Tratamiento de expresiones de cantidad. Por ejemplo, *"algo más de dos millones"*.

4. La segunda capa recupera sintagmas adverbiales y grupos verbales de primer nivel. Los sintagmas adverbiales están formados por secuencias de adverbios, solamente se tiene en cuenta el último adverbio de la secuencia como núcleo del sintagma. Aunque no existe ningún tipo de dependencia recuperada que trabaje con adverbios, este paso es necesario para el correcto funcionamiento de las etapas posteriores. Los grupos verbales de primer nivel son los sintagmas verbales no perifrásticos, correspondientes a formas pasivas, tanto simples (*"es analizado"*) como compuestas (*"ha sido analizado"*) y los sintagmas verbales de las formas activas.

5. La tercera capa detecta los sintagmas adjetivales así como los grupos verbales de segundo nivel. Un sintagma adjetival será un sintagma cuyo núcleo es un adjetivo que previamente podrá tener un adverbio que lo modifique. Los grupos verbales de segundo nivel incluyen las formas perifrásticas, conjunto de formas verbales que funcionan en unión.

6. La cuarta capa procesa los sintagmas nominales. Por definición este tipo de sintagmas tienen por núcleo un elemento que tiene una función sustantiva. Se detectan también en este punto, los sintagmas adjetivales que modifican a un sustantivo.

7. La quinta capa se encarga de la identificación de los sintagmas preposicionales, es decir los sintagmas nominales precedidos por una preposición.

10.3. Preprocesador

El primer paso en el procesamiento tanto de documentos como de consultas consiste en pre-procesar el texto para, por una parte, segmentarlo adecuadamente, y por otra, para realizar diversas transformaciones en el mismo y así facilitar el trabajo de las fases posteriores. El concepto lingüístico de palabra no siempre coincide con su realización gráfica. Por ello, Erial posee un segmentador avanzado que realiza tareas de pre-etiquetación de los textos sobre secuencias de caracteres no ambiguas

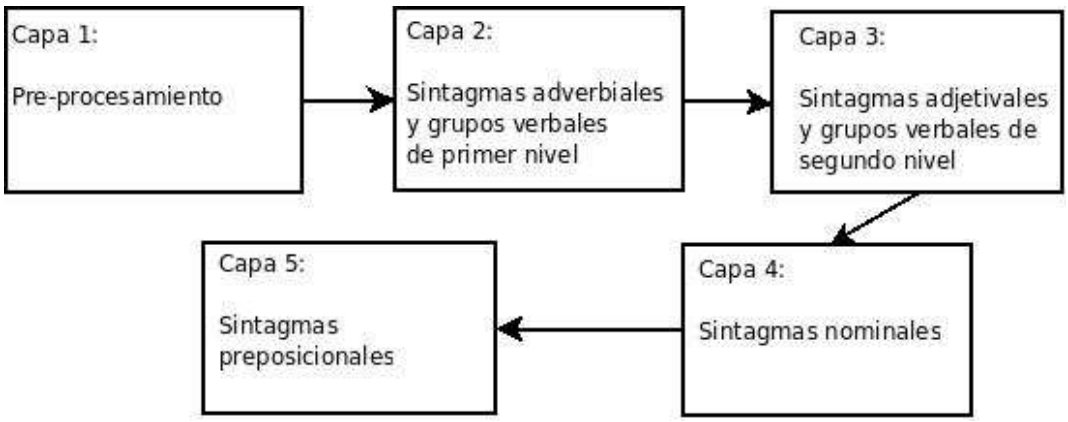

Figura 10.1: Funcionamiento de Erial

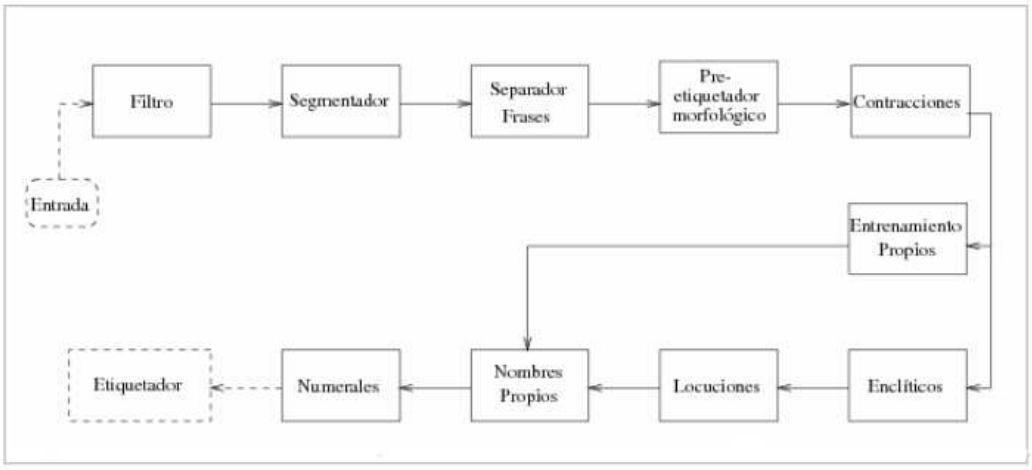

Figura 10.2: Estructura del Pre-procesador

tales como números o fechas, al tiempo que trata fenómenos como abreviaturas, acrónimos, contracciones, pronombres enclíticos, identificación de expresiones, etc.

En el caso de este proyecto se han obviado estas etapas. El propio etiquetador reconoce adecuadamente los nombres propios (por el lexicón que específicamente se creó para él) y etiqueta bien las fechas. Respecto a las contracciones y a los numerales, su tratamiento sólo se hizo necesario a nivel del análisis sintáctico (Sección: 9); ya que a nivel de adquisición de terminología la solución devuelta por *TreeTagger* resultó adecuada. Asimismo el tratamiento de *TreeTagger* respecto a las locuciones y pronombres enclíticos respondía a las necesidades planteadas en el proyecto. A continuación se describen los tres módulos que se han tomado en cuenta:

- El módulo de **filtro** es el encargado de compactar los separadores de tokens, es decir, de eliminar múltiples espacios, espacios a inicio de frase, etc. Además puede incluir cualquier otro tipo de función que facilite el desarrollo de los módulos posteriores.

- El módulo **segmentador** es el que identifica y separa las palabras presentes en el texto, de manera que cada palabra individual y también cada signo de puntuación constituyan una palabra diferente. El modulo considera las abreviaturas, las siglas, los números con decimales, o las fechas en formato numérico, para ello utiliza un diccionario de abreviaturas y otro de siglas.

- El módulo de **separador** de frase es el que delimita las frases, tarea que aunque es aparentemente sencilla, presenta numerosas dificultades. La regla general consiste en separar una frase cuando hay un punto seguido de mayúscula, pero se deben tener en cuenta las abreviaturas para no segmentar después de un punto de alguna abreviatura especial.

10.4. Etiquetador

Tras segmentar y pre-procesar el texto, el siguiente paso consiste en etiquetarlo y lematizarlo. Con ello conseguimos obtener los rasgos morfosintácticos más relevantes de cada palabra, información que será utilizada en las siguientes etapas, al tiempo que logramos eliminar la variación flexiva, normalizando las distintas realizaciones de verbos, sustantivos y adjetivos, a una sola forma, su lema. La etiquetación de las partes del discurso se realiza mediante un modelo oculto de HMM[9][10] de segundo orden.

- Características de los modelos: Los estados del HMM representan pares de etiquetas, mientras que las salidas representan las palabras. La probabilidad del mejor camino se calcula mediante el algoritmo de Viterbi, utilizando las probabilidades de transición entre trigramas. Tanto las probabilidades de transición como las de salida se estiman a partir de un corpus etiquetado de entrenamiento.

- Manejo de palabras desconocidas: Dada una palabra desconocida, se establece su conjunto de etiquetas candidatas, así como sus probabilidades asociadas, mediante el estudio de los sufijos de la palabra. La distribución de probabilidad de un determinado sufijo se genera a partir de todas las palabras del *corpus* de entrenamiento que comparten el mismo sufijo, considerando una longitud máxima predefinida.

- Integración de diccionarios externos: El modelo en el que se basa el etiquetador-lematizador soporta la integración de fuentes de información adicionales basadas en diccionarios externos, lo que permite incorporar en el modelo estadístico información adicional sin afectar al rendimiento de la etiquetación.

- Manejo de segmentaciones ambiguas: Debido a las posibles segmentaciones ambiguas detectadas por el preprocesador, el etiquetador debe ser capaz de tratar con secuencias de tokens de diferentes longitud es, lo cual implica no sólo decidir la etiqueta que les debe ser asignada, sino también determinar si algunos de ellos forman conjuntamente un sólo término o no.

10.5. Analizador sintáctico superficial

En el ámbito de la recuperación de información se denomina *término multipalabra* a aquel término que contiene dos o más palabras con contenido (sustantivos, verbos y adjetivos). Erial incorpora soporte para la extracción de este tipo de términos compuestos basada en el uso de un analizador sintáctico parcial.

En general, uno de los métodos más utilizados es la denominada simplificación del texto: se comienza con el *stemming* (obtener la raíz) de las palabras individuales junto con la eliminación de las *stopwords*, para posteriormente extraer y normalizar los términos relacionados empleando técnicas estadísticas. Existe, pues, una clara falta de base lingüística en dichas operaciones, lo que redunda frecuentemente en simplificaciones erróneas.

Existen otros métodos con base lingüística que realizan un análisis sintáctico del texto, el cual devuelve a su salida un conjunto de árboles que denotan relaciones de dependencia entre las palabras involucradas. De este modo, estructuras con relaciones de dependencia similares pueden ser normalizadas a una misma forma.

A medio camino entre ambas alternativas, está la *correspondencia de patrones*, que se basa en la hipótesis de que las partes más informativas del texto siguen unas construcciones sintácticas bastante bien definidas que se pueden aproximar mediante patrones. Esta es la aproximación seguida por Erial para implementar la extracción de términos de indexación complejos.

Las consultas realizadas a un sistema de recuperación de información pueden expresarse en forma de grupos nominales de complejidad diversa. Teniendo esto en cuenta, Erial permite tomar los grupos nominales como términos base a partir de los cuales, mediante la aplicación de las transformaciones correspondientes, se obtendrán sus variantes sintácticas y morfosintácticas, que no necesariamente serán grupos nominales.

En Erial se parte de los árboles básicos de las frases nominales y de sus variantes, transformándolos en un conjunto de expresiones regulares que serán emparejadas con el texto para extraer los pares de dependencias que se utilizarán para construir los términos índice. De esta forma, Erial identifica los pares de dependencias resultantes mediante un emparejamiento de patrones sobre la salida del *etiquetador/lematizador*, trabajando siempre con técnicas de estado finito, lo que conlleva una importante reducción en lo que respecta al tiempo de ejecución.

Capítulo 11

Apéndice B: Cadena de procesamiento lingüístico ALPAGE

11.1. Introducción

La cadena de procesamiento Alpage fue desarrollada por el equipo *Alpage*[1] con el fin de obtener una completa cadena de procesamiento lingüístico para el francés. Esta cadena está basada en los siguientes componentes:

- Sxpipe: Una cadena de pre-análisis para el francés que incluye segmentación, correcciones ortográficas, búsqueda de lexicones, detección de entidades con nombre, etc.
- DyALog: Compilador de analizadores sintácticos y entorno de programación lógica.
- Frmg: Una meta-gramática [5] [6] para el francés.
- Lefff: Lexicón morfológico y sintáctico para el francés.

Hay disponible una demo, basada en la última versión estable de la cadena, en la siguiente dirección: *http://alpage.inria.fr/parserdemo*.

[1] http://alpage.inria.fr/

11.2. Funcionamiento

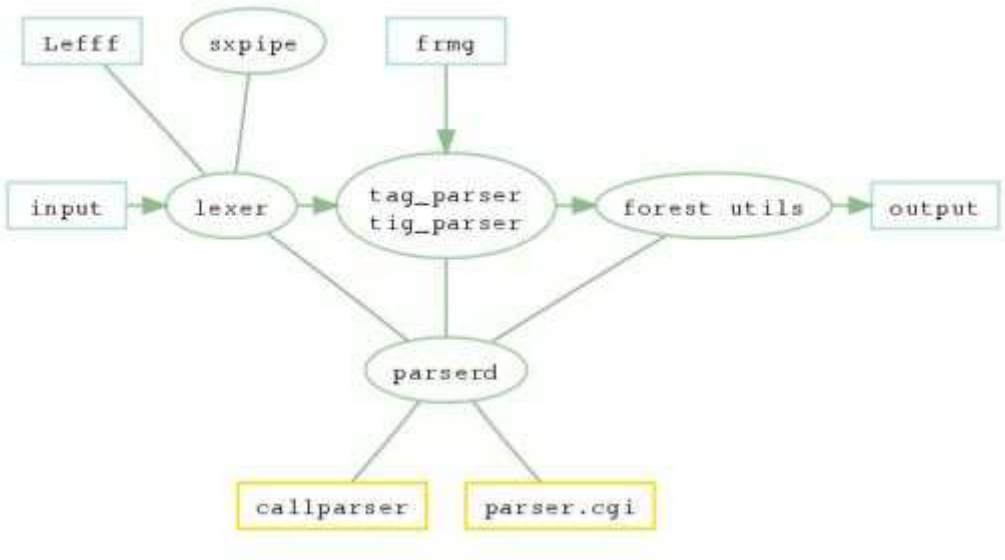

Figura 11.1: Cadena de Procesamiento Lingüístico Alpage

La figura 11.1 muestra una visión simplificada de la *Cadena de Procesamiento Lingüístico para el francés Alpage*.

El *lexer* usa *sxpipe* (una completa cadena de pre-procesamiento lingüístico, 11.2.2) para pre-procesar la entrada (frase, o archivo de frases) y combina la salida de sxpipe con información léxica recuperada de *Lefff*. La salida del lexer puede ser entonces procesada por el parser *TAG* (basado en la meta-gramática frmg). El bosque producido por el parser puede ser convertido a diversos formatos con *forest_utils*(11.2.8). El usuario puede interactuar con el servidor de parsers (*parserd*, 11.2.9), o directamente llamar a cada componente.

A continuación se describe brevemente el funcionamiento de estos componentes:

11.2.1. Lefff, lexicón sintáctico para el francés

Lefff (Lexique des Formes Fléchies du Francais - Lexicón de Formas Declinadas del Francés) es un amplio lexicón morfológico y sintáctico, cuya arquitectura depende de propiedades de herencia que lo hacen más compacto, facilitan su mantenimiento y permiten describir entradas léxicas independientemente de los formalismos para los que son usados.

11.2.2. sxpipe, una cadena de pre-procesado

Sxpipe es una completa arquitectura para el pre-procesado de texto antes del análisis sintáctico. Transforma corpus sin tratar en un *Grafo Dirigido Acíclico*

(DAG), que será usado como entrada por el analizador sintáctico. Este proceso incluye reconocimiento secuencial de entidades con nombre, tokenización y detección de límites entre frases, corrección ortográfica, procesamiento no determinista de términos multi-palabra, re-acentuación y de-/re-capitalización. Ver figura 11.2.

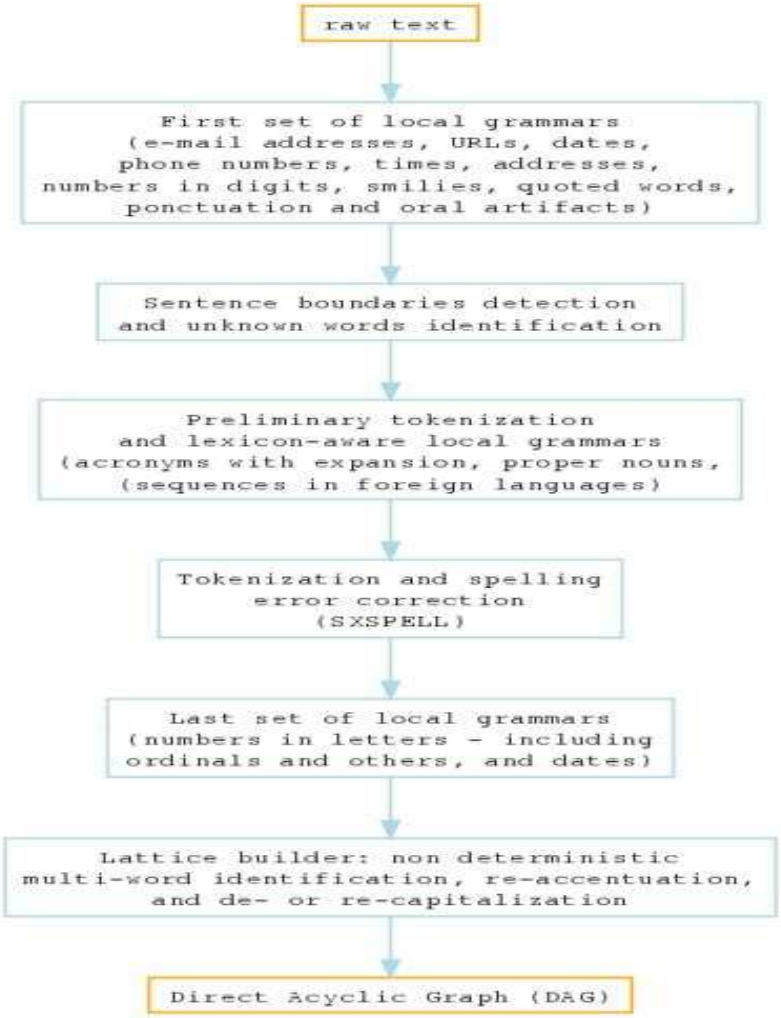

Figura 11.2: Cadena de pre-procesado sxpipe

11.2.3. El lexer easy.pl

Easy.pl, incluído en el paquete frmg, es una sencilla herramienta de segmentación que puede tomar como argumento una frase o un DAG.

11.2.4. Dyalog

DyALog [4] es un entorno usado para compilar y ejecutar programas lógicos y parsers tabulares de lenguajes naturales para diversos formalismos gramaticales (DCG's, TAG's, TIG's, RCG's). En la Cadena de Procesamiento Lingüístico Alpage, DyALog es usado para producir dos parsers basados en la meta-gramática frmg.

11.2.5. Meta-gramática francesa frmg

La Meta-Gramática es una gramática abstracta diseñada para ser compilada en TAG. Se puede encontrar la meta-gramática original en el siguiente archivo:

$PREFIX/share/frmg/frgram.smg

11.2.6. De la mega-gramática a los parsers

La meta-gramática (MG) francesa frmg viene en formato .smg y sufre una serie de transformaciones para producir un parser:

1. frmg es compilada en formato xml con la herramienta smg2xml
2. Esto se convierte a formato DyAlog (mg.pl) gracias a la herrmaienta mg2dyalog
3. El resultado se compila con mgcomp (MG compiler) a una representación XML (TAGXML) de una TAG (.tag.xml).
4. La gramática se convierte en formato DyALog (.tag) con tag_converter.
5. Entonces es compilado a un TAG o TIG/TAG parser con dyacc, incluído en el paquete DyALog.

11.2.7. Los parsers tig_parser y tag_parser

Estos parsers son generados por DyALog a partir de la meta-gramática frmg.

11.2.8. El conversor forest

Forest_convert.pl es un script perl contenido en el paquete forest_utils. Permite convertir los bosques producidos por los parsers TAG a diversos formatos más útiles.

11.2.9. parsed, un servidor de parsers

Parserd tiene dos clientes: parser.pl y callparser. Hay tres posibilidades para comunicarse con el servidor:

- A través de la interfaz *web* proporcionada por parser.pl.
- Con el comando *callparser*, que usa *telnet* para comunicarse con *parserd*.
- Usando *telnet* directamente.

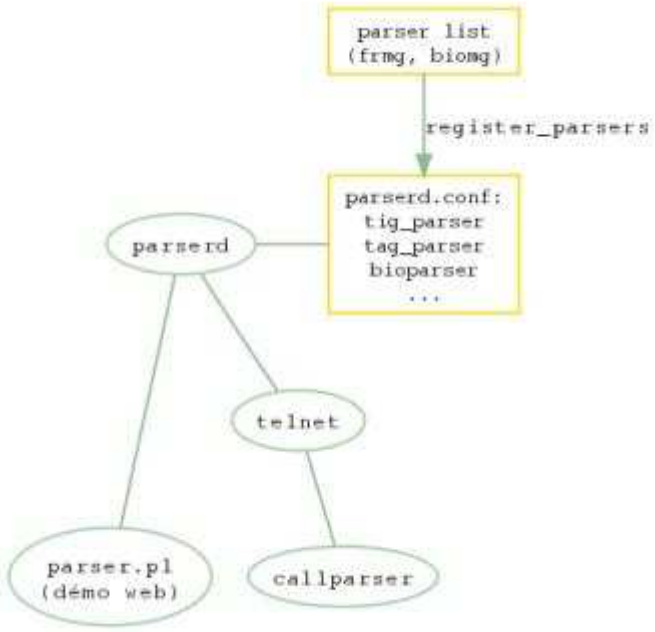

Figura 11.3: Servidor de parsers

11.3. Instalación

La forma más fácil de instalar la Cadena de Procesamiento Lingüístico Alpage es usar el instalador Alpage (alpi.pl). *alpi* es un *script Perl* que ayuda a los usuarios a instalar localmente software desarrollado por el equipo del proyecto Alpage. Este script consta de tres pasos. Para cada paquete de la cadena:

- Descarga las fuentes.
- Detecta e instala dependencias.
- Instala el paquete.

11.3.1. Descargar el instalador Alpage

Descargar desde la siguiente dirección y descomprimir:

http://alpage.inria.fr/ cabrera/download/alpi.pl.zip.

11.3.2. ¿Qué se necesita?

Antes de ejecutar el instalador alpi es necesario tener instalados los siguientes paquetes y/o servicios:

- Conexión a Internet (el *script* obtiene los paquetes a través de FTP o Subversion).
- Perl 5.8 o superior.

- AppConfig
- IPC::Run
- Pkg-config 0.15.0.
- g++.
- perlcc: para Lefff (perlcc es un compilador de Perl a código máquina, parte del paquete debían/ubunti libperl-dev o del paquete mandriva perl-devel.rpm).
- recode: para parserd.
- Xsltproc: para frmg.
- Bison > 2.3: para DyALog.
- Flex: para DyALog.
- telnet: para parserd.

Si se obtienen las fuentes a través del repositorio *Subversion*:

- Subversion.
- Autoconf 2.60 o superior.
- Automake 1.10 o superior.
- Libtool 1.5 o superior.
- Mekeinfo: para DyALog.

Otras dependencias opcionales pero recomendadas:

- *ImageMagick*: para mostrar un gráfico de dependencias (comando display).
- *Dillo*: para visualizar flujos HTML producidos por callparser.
- *uDraw* (originalmente conocido como daVinci): visualiza jerarquías de meta-gramáticas con mgviewer (paquete mgtools).

subsectionInstalación para usuarios *./alpi.pl*
Este comando instala la Cadena de Pre-procesamiento Lingüístico descargando los paquetes en formato .tar.gz a través de FTP.
Nota: No es recomendable ejecutar alpi como root porque la cadena se instala localmente. En futuros desarrollos será posible instalar como root. Por defecto la cadena se instala en el directorio $HOME/exportbuild.

11.3.3. Opciones

- config=file: para leer un fichero de configuración específico (por defecto se lee alpi.conf).
- prefix=path (por defecto /home/usuario/exportbuild).
- vardir=path (por defecto /tmp/usuario/exportbuild).
- port—p=< *numero* > (por defecto 8999)
- user—u=< *usuario* >
- group—g=*grupo* >
- svn=< $gforge - login$ >: para descargar las fuentes desde el repositorio svn de GForge.
- package—pkg=< *paquete* >: para descargar un paquete en concreto.
- skippkg=< *paquete* >: para saltarse la instalación de un paquete en concreto.
- Skipdep=< *paquete* >: para saltarse la instalación de todas las dependencias.
- linux32: para forzar la emulación del modo 32bit en máquinas de 64 bits.
- force—f
- dependencies—dep: para mostrar un listado de todas las dependencias necesarias para ejecutar el script.
- version—v: para mostrar por pantalla las versiones de los paquetes Alpage.
- project: para mostrar el nombre del proyecto GForge de cada paquete.
- log=< $fichero_de_log$ > (por defecto alpi.log).
- test: ejecuta tests para los paquetes (por defecto no lo hace).

11.4. Tutorial

11.4.1. Configurar el entorno

Para usar la cadena directamente, es necesario, en primer lugar, configurar las variables de entorno mediante el uso del siguiente comando:

source $PREFIX/sbin/setenv.sh

Donde $PREFIX es el directorio donde se ha instalado la cadena (por defecto $HOME/exportbuild). Esta configuración es válida solamente para la sesión actual. Para conservarla permanentemente es necesario añadir el contenido de setenv.sh en el fichero .bashrc.

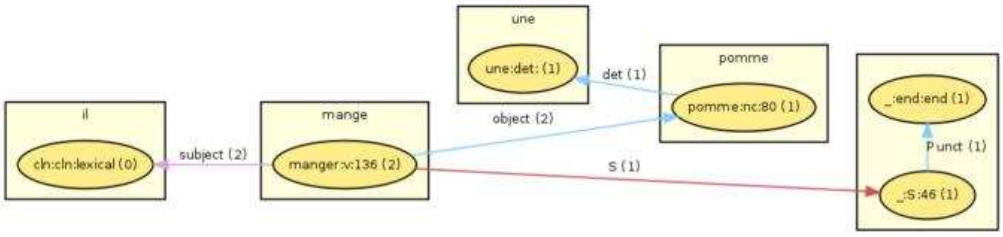

Figura 11.4: Vista gráfica de las dependencias

11.4.2. Uso de los analizadores sintácticos

El servidor de analizadores sintácticos *parserd* pude ser iniciado y parado por medio de los siguientes comandos:

$PREFIX/sbin/parserd_setenv start

$PREFIX/sbin/parserd_setenv stop

El servidor de analizadores sintácticos puede ser accedido a través del comando *callparser*. Para visualizar las dependencias gráficamente:

echo "Il mange une pomme" | callparser -in - -d dep

Nota: Para poder mostrar este gráfico de dependencias es necesario tener instalados en el sistema *Graphviz* e *ImageMagick*. Para conseguir una representación en formato XML de las dependencias:

echo "il mange une pomme" | callparser -in - -xmldep

Para obtener estadísticas de tiempo y de ambigüedad :

echo "Il mange une pomme" | callparser -in -time -stats

Para analizar un fichero de frases (una frase por línea):

cat < fichero > | callparser -in - -xmldep

Para almacenar una colección de frases analizadas en un directorio:

cat < fichero > | callparser -in - -collsave -xmldep -collbase sentence -colldir mycoll

callparser tiene muchas opciones para visualizar la salida (HTML, XML ...). Para encontrar más información sobre el uso de *callparser*:

Perldoc callparser

11.4.3. Resolución de problemas

En caso de encontrarse con algún problema es recomendable visitar el siguiente enlace: *http://alpage.inria.fr/alpchainfaq.en.html* De todas maneras, hay una serie de comandos que nos permiten rastrear o solucionar temporalmente el problema: Revisar la salida de *easy.pl*:

echo "Je teste une phrase" | easy.pl

Si obtenemos algo similar a lex => '_uw'significa que se ha encontrado una palabra desconocida o que alguna palabra ha sido etiquetada con una categoría errónea. Es posible solucionar esto editando missing.lex, en el directorio $PREFIX/share/frmg/. Para revisar directamente una entrada del lexicón:

echo "Luc" — lexed -d $PREFIX/lib/lefff-frmg -p dico , xlfg consult

Se puede encontrar más información sobre el uso de lexed mediante el siguiente comando:

lexed -usage

Bibliografía

[1] Booch, Grady Rumbaugh, James Jacobson, Ivar. *"El Lenguaje Unificado de Modelado"*. 2000.

[2] Kruchten, Philippe *"The Rational Unified Process: An Introduction"*. 2003.

[3] Lamport, Leslie *"LaTeX: A Document Preparation System"*. 1986.

[4] De la Clergerie, E. *"Dyalog: a tabular logic programming based environment for NLP"*. In Proc. of 2nd Int. Workshop on Constraint Solving and Language Processing, Barcelona, Spain 2005.

[5] De la Clergerie, E. *"From metagrammars to factorized TAG/TIG parsers"*. In Proc of IWPT '05 pages 190-192, Vancouver Canada oct 2005

[6] *"Tabular algorithms for TAG parsing"*. In proc. of EACL'99. Ninth Conf of European Chapter of the Association for Computational Linguistics, P 150-157, Norway, 1999.

[7] *"Extracción de términos índice mediante cascadas de expresiones regulares"*. In Actas de las II Jornadas de Tratamiento y Recuperación de Información (JOTRI 1003), pp. 204-211, Leganés (Madrid), Spain, 2003.

[8] Graña, J. *"Técnicas de Análisis Sintáctico Robusto para la Etiquetación del Lenguaje Natural"*. PhD thesis, Departamento de Computación, Universidade da Coruña, Diciembre 2000.

[9] Baum, L.E, Petrie, T. *"A maximization technique ocurring in the statistical analysis of probabilistic function of Markov"*. 1970.

[10] Baum, L., Petrie, T. *"Statistical inference for probabilistic functions of finite state Markov chains"*. Annals of Mathematical Statistics, 41, 1996.

[11] Graña, J., Chappelier, J., Vilares, M. *"Integrating external dictionaries into stochastic part-of-speech taggers"*. In EuroConference Recent Advances in Natural Language Processing. Proceedings, pp. 122-128, Tzigov Chark, Bulgaria, 2001.

[12] De la Clergerie, E. *"MAF: a Morphosyntactic Annotation Framework"*. In proc. of the 2nd Language and Technology Conference (LT'05), pp. 90-94, Poznan, Poland, April 2005.

[13] Vilares Ferro, J. Barcala, F. M. Alonso, M. Graña, J. Vilares, M. *"Practical NLP-Based Text Indexing"*. In proc. of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence, pp. 635-644, 2002.

Índice alfabético

alpage, 19, 77, 133
análisis sintáctico, 17, 19, 26, 36, 61, 95
analizador sintáctico, 18, 19, 26, 53, 74, 93, 96, 113

cluster, 21

dependencia, 20, 76, 120

erial, 19, 76, 127

frase, 20, 26, 51, 54, 75

java, 22, 33, 75, 105

lenguaje, 20, 98
    natural, 17, 18, 46

nodo, 21

recuperación de información, 17, 19, 46

texto, 26, 51, 54, 75

uml, 29, 38

xml, 19, 26, 34, 54, 95