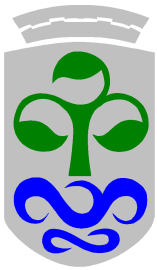

UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑERÍA INFORMÁTICA

Memoria do Proxecto Fin de Carreira que presenta

D. Mª Nieves Fernández Formoso

para a obtención do Título de **Enxeñeiro en Informática**

Implementación de una librería para el manejo eficiente de diccionarios de gran tamaño

Xaneiro, 2009

Proxecto Fin de Carreira Nº: ENI-235

Director/a: Francisco Mario Barcala Rodríguez
Área de coñecemento: Ciencias da Computación e Intelixencia Artificial
Departamento: Informática

"El destino es el que baraja las cartas, pero nosotros los que las jugamos."

Arthur Schopenhauer

A mis padres Andrés y Estrella.

AGRADECIMIENTOS

Es el momento de expresar mi gratitud a todas las personas que me han ayudado a finalizar esta importante etapa en mi vida.

En primer lugar a mi director de proyecto, Francisco Mario Barcala Rodríguez, su paciencia y comprensión han hecho posible superar los obstáculos encontrados y su constante ayuda ha permitido alcanzar la conclusión del trabajo.

A mis compañeros de laboratorio, por su apoyo y por los buenos momentos que hemos pasado: Adrián, Juan, Fran, Víctor M., Mila, Erica y Sara y especialmente a Manuel Vilares por la oportunidad dada y la confianza que ha depositado en mí.

A mi familia, especialmente a mis padres Andrés y Estrella por estar siempre a mi lado, por cada palabra, por cada gesto, soy lo que soy gracias a ellos.

Este largo camino no lo he recorrido sola, siempre que he tropezado me han ayudado a levantarme, Tania y Natalia que puedo deciros, significáis para mi más de lo que puedo expresar con palabras. Y finalmente a ti, por estar siempre ahí, Víctor sabes que sin ti no hubiera llegado hasta aquí.

A todos y cada uno, gracias.

TABLA DE CONTENIDO

MEMORIA ---- **11**

IDENTIFICACIÓN DEL PROYECTO ---- 13
ESTADO DEL ARTE ---- 13
ORGANIZACIÓN DE LA DOCUMENTACIÓN ---- 15
DESCRIPCIÓN DEL PROYECTO ---- 16
DESARROLLO DEL PROYECTO ---- 17
1. OBJETIVOS GENERALES DEL SISTEMA ---- 17
2. DESCRIPCIÓN GENERAL DE LA ARQUITECTURA ---- 18
3. DATOS TÉCNICOS DEL PROYECTO ---- 20
4. PLANIFICACIÓN TEMPORAL ---- 20
4.1. PLANIFICACIÓN PREVIA ---- 20
4.2. PLANIFICACIÓN REAL ---- 22
PRESUPUESTO ---- 23
1. RECURSOS FÍSICOS ---- 23
2. RECURSOS HUMANOS ---- 24
3. PRESUPUESTO TOTAL ---- 25
CONCLUSIONES ---- 25
1. AMPLIACIONES ---- 26

MANUAL TÉCNICO ---- **27**

INTRODUCCIÓN ---- 29
1. DOCUMENTACIÓN TÉCNICA ---- 29
2. SISTEMA IMPLEMENTADO ---- 29
2.1 Compilador ---- 31
2.2 Librería de acceso ---- 32
3. DICCIONARIO DE EQUIVALENCIAS ---- 32
ANÁLISIS ---- 34
1. INTRODUCCIÓN ---- 34
2. DESCRIPCIÓN ---- 34
3. ANÁLISIS DE REQUISITOS ---- 34
4. CASOS DE USO ---- 36
4.1. CASO DE USO: COMPILACIÓN DICCIONARIO ---- 36
4.2. CASO DE USO: GESTIÓN DICCIONARIO COMPILADO ---- 40
4.2.1 ESCENARIO: CARGAR DICCIONARIO COMPILADO EN MEMORIA ---- 40
4.2.2 ESCENARIO: CONVERSIÓN PALABRA A ÍNDICE ---- 42
4.2.3 ESCENARIO: CONVERSIÓN ÍNDICE A PALABRA ---- 44
4.2.4 ESCENARIO: ACCEDER INFORMACIÓN COMPLETA DE PALABRA ---- 46
4.2.5 ESCENARIO: ACCEDER INFORMACIÓN CONCRETA DE PALABRA ---- 48
4.2.6 ESCENARIO: ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA ---- 50
4.2.7 ESCENARIO: ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA ---- 52
4.2.8 ESCENARIO: LIBERAR RECURSOS ---- 54
5. DIAGRAMA DE CLASES ---- 56
DISEÑO ---- 56
1. DIAGRAMA DE CLASES ---- 56
2. DICCIONARIO DE CLASES ---- 58
2.1 Compilador ---- 58

2.2 NADFA ---- *60*
2.2.1 NADFA_Automata ---- 62
Construcción ---- 63
Escritura ---- 73
2.2.2 NADFA_Mappings ---- 76
Construcción ---- 77
Escritura ---- 78
2.2.3 NADFA_Infos ---- 79
Construcción ---- 81
Escritura ---- 82
2.3 NADFA_BIN ---- *84*
2.4 Entrada/Salida ---- *91*
3. DIAGRAMAS DE DISEÑO PARA LOS CASOS DE USO DEL SISTEMA ---- *92*
3.1. CASO DE USO: COMPILACIÓN DICCIONARIO ---- *92*
3.2. CASO DE USO: GESTIÓN DICCIONARIO COMPILADO ---- *95*
3.2.1 ESCENARIO: CARGAR DICCIONARIO COMPILADO EN MEMORIA ---- 95
3.2.2 ESCENARIO: CONVERSIÓN PALABRA A ÍNDICE ---- 96
3.2.3 ESCENARIO: CONVERSIÓN ÍNDICE A PALABRA ---- 98
3.2.4 ESCENARIO: ACCEDER INFORMACIÓN COMPLETA DE PALABRA ---- 99
3.2.5 ESCENARIO: ACCEDER INFORMACIÓN CONCRETA DE PALABRA ---- 101
3.2.6 ESCENARIO: ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA ---- 102
3.2.7 ESCENARIO: ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA ---- 104
3.2.8 ESCENARIO: LIBERAR RECURSOS ---- 105
IMPLEMENTACIÓN ---- 107
1. ARQUITECTURA DEL SISTEMA ---- *108*
2. DIAGRAMA DE COMPONENTES ---- *109*
3. FICHERO XML DE CONFIGURACIÓN ---- *109*
PRUEBAS ---- 112
1. Rendimiento ---- *113*
1.1 Compilación ---- *113*
1.2 Tasa de Reconocimiento ---- *113*
2. Estabilidad ---- *114*
MANUAL USUARIO ---- 121
INTRODUCCIÓN ---- 123
REQUISITOS ---- 123
1. REQUISITOS HARDWARE ---- *123*
2. REQUISITOS SOFTWARE ---- *124*
COMPILACIÓN ---- 124
EJECUCIÓN Y FUNCIONAMIENTO DE LA LIBRERÍA ---- 125
1. Fichero de configuración XML ---- *126*
BIBLIOGRAFÍA ---- 131
TABLA DE FIGURAS ---- 133

MEMORIA

IDENTIFICACIÓN DEL PROYECTO

Características del proyecto.

Código del PFC: **ENI-235.**

Título del PFC: **Implementación de una librería para el manejo eficiente de diccionarios de gran tamaño.**

Alumna: **Mª Nieves Fernández Formoso.**

Director del proyecto: **Francisco Mario Barcala Rodríguez.**

Departamento: **Informática.**

Área: **Ciencias de la computación e inteligencia artificial.**

Titulación: **Ingeniería Informática.**

ESTADO DEL ARTE

Aspectos generales del proyecto.

En la actualidad existe un gran volumen de información textual disponible en formato digital que hace necesario el desarrollo de herramientas que permitan su gestión, recuperación y procesamiento. Para atender a estas necesidades existen diferentes técnicas y/o algoritmos que se centran principalmente en dos aspectos: eficiencia y precisión.

Las aplicaciones que se basan en la utilización de esta gran cantidad de información son de muy diversa índole: procesamiento automático, clasificación, recuperación de información, consulta, etc. pero este proyecto se centrará, principalmente, en las aplicaciones relacionadas con el procesamiento del lenguaje natural, en adelante PLN.

El PLN se engloba dentro de los campos de la Inteligencia Artificial y la Lingüística Computacional para tratar, como su último fin, los problemas de la generación automática del lenguaje y el entendimiento por parte de computadores del lenguaje humano.

Dado que aún se está lejos de tratar directamente este objetivo final, en el PLN se identifican múltiples y diferentes subproblemas o subcampos de trabajo: reconocimiento de voz, segmentación de texto, desambigüación del sentido de las palabras, análisis sintáctico de las oraciones, identificación del significado de las oraciones, generación de resúmenes, respuesta

a preguntas, generación del lenguaje, recuperación de información, etc. que necesitan del desarrollo de técnicas que actúen en los siguientes niveles del lenguaje:

Nivel morfológico: Engloba el estudio y/o tratamiento de los problemas relacionados con la morfología de las palabras.

Nivel léxico: Estudio y/o tratamiento de las palabras de un modo individual.

Nivel sintáctico: Trata las relaciones entre las palabras, o lo que es lo mismo, las palabras dentro de su contexto de aparición.

Nivel semántico: Estudia el significado del texto.

Nivel pragmático: Trata la interpretación de la semántica asociándola con la realidad a la que se refiere.

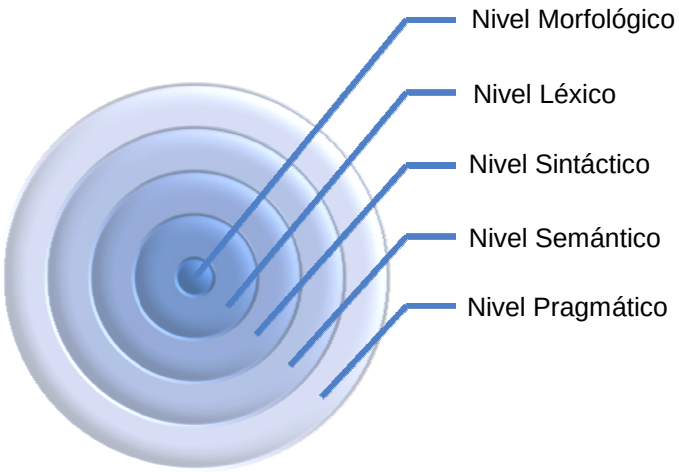

Figura 1. Niveles del lenguaje.

Este proyecto se centra en los niveles morfológico y léxico y, en particular, en aplicaciones que requieren el uso de diccionarios de grandes dimensiones, entendiendo por diccionario cualquier estructura que permita acceder a cualquier tipo de información asociada a las palabras en ella almacenada.

La tecnología más ampliamente utilizada para la materialización de un diccionario es la de las bases de datos donde, por ejemplo, se puede crear una tabla donde la clave sean las palabras y las diferentes columnas almacenen la información asociada a las mismas. Sin embargo, aunque las bases de datos son muy flexibles y permiten almacenar prácticamente cualquier tipo de información, resultan muy lentas en los casos en que es necesario un procesamiento en tiempo real.

Es el caso, por ejemplo, de la etiquetación automática, o lo que es lo mismo, la desambigüación de las categorías gramaticales de las palabras. Esta tarea, en muchas ocasiones, forma parte de procesos más complejos y mucho más ambiciosos que incluyen

tareas relacionadas con niveles superiores al morfo-léxico, como puede ser el análisis sintáctico de una consulta para su utilización en la recuperación de información, por lo que el acceso a una base de datos para el acceso a la información asociada a las palabras puede resultar demasiado costoso, o incluso inservible en muchos casos.

Es por ello necesario el desarrollo de otras implementaciones de diccionarios que garanticen un rendimiento óptimo en el acceso a las palabras para que no penalicen el tiempo necesario para resolver tareas de más alto nivel. Es el caso de, por ejemplo, los autómatas finitos (1), que constituyen una alternativa mucho más eficiente que las bases de datos en diversas tareas relacionadas con el PLN.

Existen trabajos donde se detallan propuestas de estructuras para el manejo de palabras utilizando autómatas finitos acíclicos deterministas numerados (2), e incluso hay alguna propuesta de estructura de diccionario utilizando autómatas (3), pero las implementaciones asociadas a estos trabajos son muy limitadas. En el primer caso solo se permite la gestión y manejo de las palabras (sin información asociada) y en el segundo la información que se gestiona es muy limitada y para un uso muy concreto: las palabras del diccionario deben de estar ordenadas, la información asociada se limita a dos columnas de información ya determinadas (no modificables), etc.

La idea de este trabajo es combinar los fundamentos de construcción de autómatas finitos mínimos acíclicos deterministas numerados propuestos por Daciuk (2), que minimizan el uso de memoria para la construcción de dichos autómatas, con la idea de construcción de diccionarios presentada por Graña (3) para implementar una librería genérica de utilización y gestión de diccionarios que minimice, tanto el uso de memoria como el tiempo de ejecución, al tiempo que permita el almacenamiento de cualquier tipo de información asociada a las palabras.

ORGANIZACIÓN DE LA DOCUMENTACIÓN

Documentos que componen el proyecto.

El CD-ROM de la aplicación tiene 2 carpetas en su interior:

- **Software:** incluye todos los ficheros que componen el sistema desarrollado.

- **Documentación:** documentación generada a lo largo del ciclo de vida del proyecto, organizada en 3 partes:
  a) **Memoria (documento actual):** en este apartado se hace una descripción del proyecto con el fin de ayudar al usuario a entender el contenido de la documentación, además de su finalidad y su proceso de desarrollo.

Para ello se describen los objetivos fundamentales del sistema y la arquitectura, se realiza una planificación temporal, el cálculo de los costes de desarrollo, conclusiones y posibles ampliaciones.

b) **Manual Técnico:** es la parte donde se documenta todo lo referente al análisis, diseño, implementación y pruebas:
   - **Análisis:** en esta parte se hace una especificación de requisitos y se muestran todos los diagramas correspondientes a dichas especificaciones, tales como: diagrama de casos de uso, diagrama de secuencia, diagrama de actividades…
   - **Diseño:** en este apartado se realiza un refinamiento de los diagramas incluidos en el análisis, obteniendo un mayor nivel de especificación del sistema.
   - **Implementación del sistema:** en esta parte se incorporan todos aquellos detalles de implementación considerados relevantes.
   - **Pruebas del sistema:** finalmente se incorporan las pruebas llevadas a cabo para comprobar el correcto funcionamiento del sistema desarrollado.

c) **Manual de usuario:** documento creado para ser la guía de instalación y puesta en funcionamiento de la librería. Para ello se explican todos los pasos a seguir para poder utilizar correctamente sus funciones.

DESCRIPCIÓN DEL PROYECTO

¿En qué consiste este proyecto?

La finalidad de este proyecto es desarrollar una librería que permita la gestión de diccionarios de grandes dimensiones de muy diversa índole utilizando, para ello, autómatas finitos acíclicos deterministas numerados. Para el desarrollo de esta librería se han seguido dos máximas:

- Minimizar el consumo de memoria necesario para los diccionarios.
- Minimizar el tiempo de ejecución para que puedan ser utilizados en sistemas que necesiten un rendimiento elevado.

La librería está formada por dos partes, una utilizada para construir el compilador encargado de compilar los diccionarios de palabras, y la otra responsable de facilitar el acceso y el uso de los diccionarios ya compilados. Lo que se necesita para compilar los diccionarios es una lista de palabras junto con la información asociada a ellas, y el resultado es un diccionario compilado (comprimido) en formato binario. Después es la segunda parte de la librería la que permite gestionar el acceso a los diccionarios compilados.

Entre las funcionalidades que tiene esta librería frente a otros trabajos e implementaciones ya existentes destacan:

- Posibilidad de incluir cualquier tipo de información asociada a las palabras.
- Capacidad para gestionar diversos diccionarios simultáneamente y con referencias entre ellos.

La construcción del autómata empleado como almacenamiento de las palabras se basa en el algoritmo de construcción de autómatas propuesto por Daciuk (2). De este modo, el autómata se construye de manera incremental y mínima, optimizando así el uso de memoria.

Para el almacenamiento de información se utilizan los planteamientos presentados por Graña (3) generalizando algunos aspectos: almacenamiento de cualquier tipo de información asociada a las palabras, eliminación de límites en el número de campos de información, posibilidad de utilizar más de un tablero de conversión (*mapping*) y posibilidad de acceder a la información en disco y/o en memoria.

DESARROLLO DEL PROYECTO

Objetivos, descripción, datos técnicos y planificación del desarrollo del proyecto.

1. OBJETIVOS GENERALES DEL SISTEMA

Para alcanzar el fin del presente proyecto y poder así desarrollar una librería que permita el manejo eficiente de diccionarios de grandes dimensiones es necesario alcanzar los siguientes objetivos:

- Construir un compilador capaz de comprimir diccionarios empleando el menor espacio de memoria y en el menor tiempo posible sin necesidad de que las palabras del diccionario estén ordenadas alfabéticamente.
- Diseñar el modelo de datos que pueda almacenar los datos del diccionario y dar soporte a las operaciones del proceso de compilación.
- Conseguir que el número de columnas de información asociadas a las palabras del diccionario no influya en la rapidez del sistema.
- Generalizar el proceso para cualquier tipo de datos de la información asociada a las palabras, minimizando al máximo la memoria empleada para su almacenamiento.
- Permitir que un diccionario pueda acceder a otros diccionarios previamente compilados.
- Localizar la posición de una palabra en el menor tiempo posible ya sea una palabra con una sola información asociada o con varias.
- Acceder a la información asociada a las palabras en el menor tiempo posible y con un consumo mínimo de memoria.

- Utilizar software gratuito para el desarrollo y la implantación del sistema.
- Realizar un diseño de la aplicación que maximice la portabilidad y la escalabilidad del sistema.

2. DESCRIPCIÓN GENERAL DE LA ARQUITECTURA

Como ya se ha comentado en la descripción inicial, el proyecto tiene dos partes bien diferenciadas. Por un lado un compilador que, dada una lista de palabras con su respectiva información y, opcionalmente, algún diccionario ya compilado que se necesite para generar el actual, genera en disco la versión binaria compilada del mismo. Los datos que necesita el compilador son proporcionados a través de un fichero XML (4) que especifica las fuentes de datos y el fichero resultante. A continuación se muestra una figura con la descripción gráfica que describe este proceso:

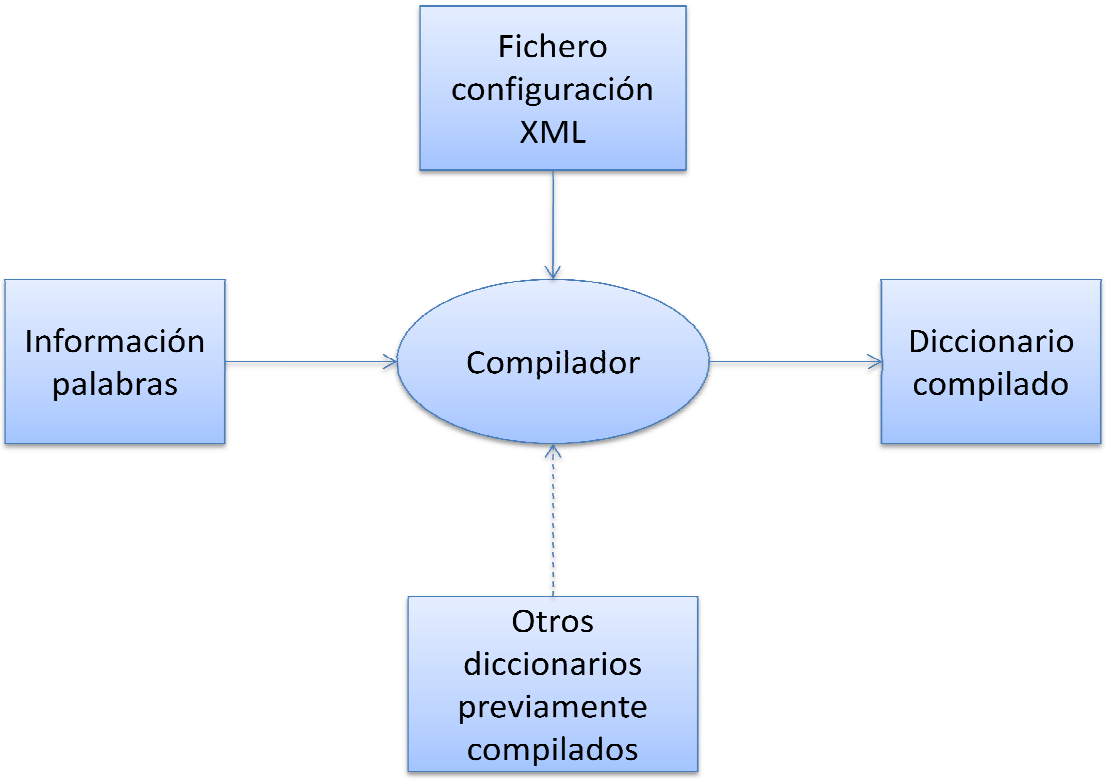

Figura 2. Arquitectura del proceso de compilación.

La estructura interna que maneja el compilador para generar el diccionario compilado es la que se muestra en la siguiente figura:

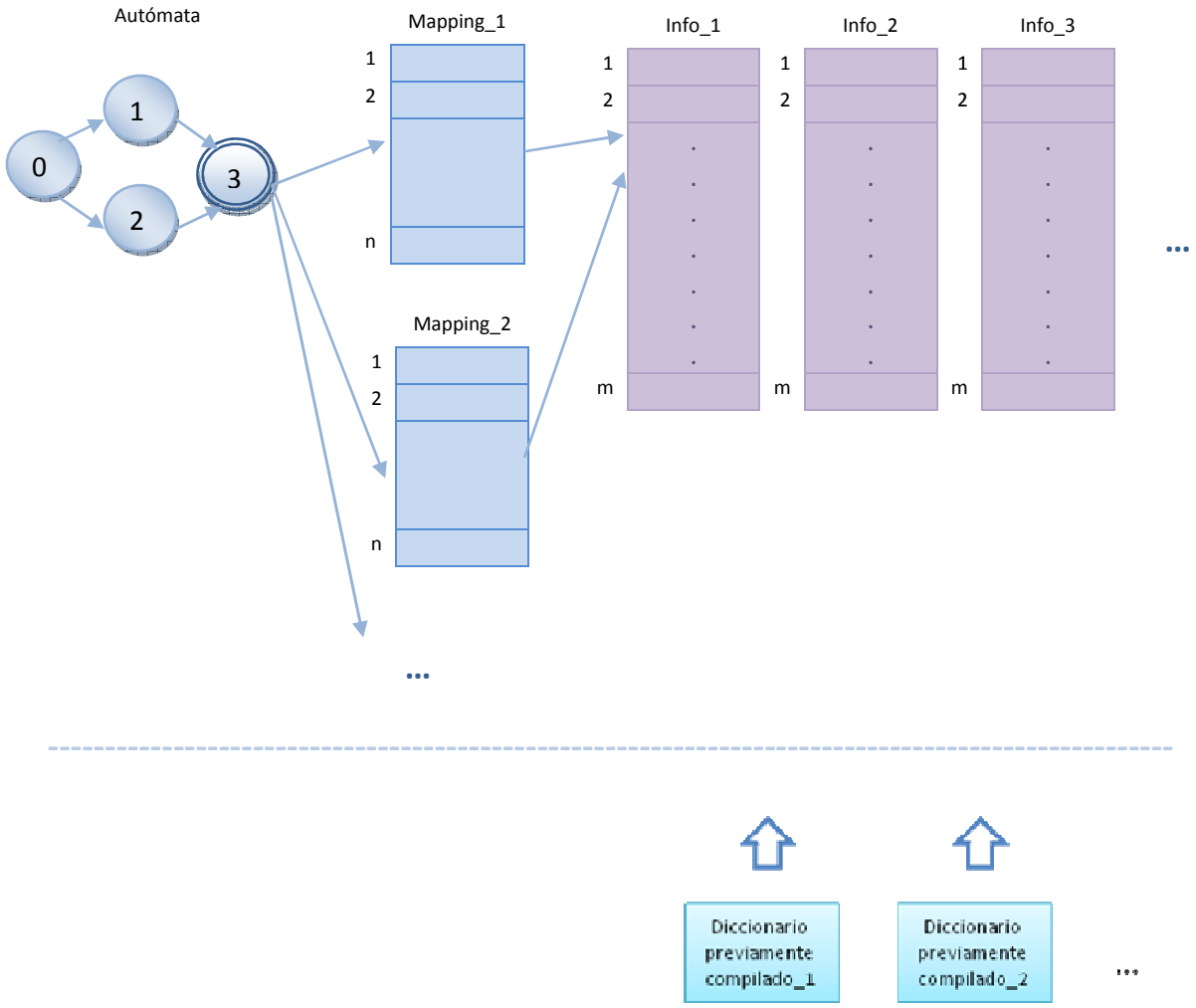

Figura 3. Estructura interna del sistema.

La segunda parte de la librería se encarga del uso y manejo de la versión compilada de los diccionarios, proporcionando las funciones necesarias para que sea útil en diferentes aplicaciones. La estructura interna que maneja esta librería de acceso sigue un esquema muy similar al presentado en la figura anterior para el compilador, con la salvedad de que utiliza la versión compilada del diccionario y, por lo tanto, estructuras muy optimizadas para el acceso a la información y no para la gestión y transformación que se necesitan durante la compilación del diccionario.

3. DATOS TÉCNICOS DEL PROYECTO

A continuación se detalla la estructura técnica del proyecto:

1. Para la implementación de la librería se utiliza el lenguaje de programación C (5), para garantizar su ejecución en cualquier entorno C o C++ (5) y poder tener un control absoluto de la gestión de memoria y ejecución (algo imprescindible para este proyecto), las librerías glib (6), para asegurar la portabilidad entre máquinas, y libxml2 (7), para el acceso al archivo XML de configuración del compilador de diccionarios.

2. Se emplea programación estructurada para organizar adecuadamente los tipos de datos y las funciones. Para el análisis y el diseño se utiliza UML (8) (9) que, si bien está más pensado para el paradigma de la programación orientada a objetos, lo utilizamos igualmente aunque desarrollemos con programación estructurada. En el manual técnico queda patente la correspondencia entre los modelos UML y las estructuras de programación estructuradas.

3. La arquitectura sobre la que se ha realizado el desarrollo y sobre la que se ha potenciado el correcto funcionamiento de la librería es GNU/Linux, garantizando un entorno robusto, estable y rápido. Esto no descarta que, si glib y libxml2 funcionan correctamente en otras plataformas, la librería tampoco debería tener problemas de funcionamiento en estos otros entornos.

4. PLANIFICACIÓN TEMPORAL

4.1. PLANIFICACIÓN PREVIA

En la planificación inicial se definieron las siguientes fases:

- Estudio del problema.
- Estudio de las herramientas a utilizar.
- Estudio de las tecnologías a utilizar.
- Análisis y diseño del sistema.
- Implementación del sistema.
- Pruebas del proyecto.

Paralelamente a todas las fases se desarrolla la documentación del proyecto. La secuencia en la que se describen las fases en la lista anterior, no implica que tenga que estar totalmente terminada una fase para pasar a la siguiente, pues existen iteraciones en el ciclo de desarrollo en función de las nuevas necesidades encontradas.

Dedicación semanal prevista (en horas/semana): 35	
Fase	**Estimación temporal** (en semanas)
Estudios previos	**3**
Estudio del problema propuesto	**1**
Investigación funcionamiento autómatas finitos acíclicos	**2**
Fase de análisis y diseño	**4**
Análisis de la aplicación	**2**
Diseño de la aplicación	**2**
Fase de implementación	**16**
Estudio de las tecnologías	**3**
Desarrollo del sistema	**13**
Puesta en funcionamiento	**1**
Pruebas finales	**2**
Documentación	**25,4**
Documentación del problema propuesto	**1**
Documentación de la investigación	**2**
Documentación del análisis	**3,6**
Documentación del diseño	**4,2**
Manual técnico y de usuario	**16,2**
TOTAL PROXECTO	**25,4**

Figura 4. Estimación temporal inicial.

	Nombre de tarea	Duración	Comienzo	Fin
1	**- PROYECTO**	**127 días**	**lun 03/03/08**	**mar 26/08/08**
2	**- Estudios previos**	**15 días**	**lun 03/03/08**	**vie 21/03/08**
3	Estudio del problema propuesto	5 días	lun 03/03/08	vie 07/03/08
4	Investigación funcionamiento autómatas finitos acíclicos	10 días	lun 10/03/08	vie 21/03/08
5	**- Fase de análisis y diseño**	**20 días**	**lun 24/03/08**	**vie 18/04/08**
6	Análisis de la aplicación	10 días	lun 24/03/08	vie 04/04/08
7	Diseño de la aplicación	10 días	lun 07/04/08	vie 18/04/08
8	**- Fase de implementación**	**80 días**	**lun 21/04/08**	**vie 08/08/08**
9	Estudio de las tecnologías	15 días	lun 21/04/08	vie 09/05/08
10	Desarrollo del sistema	65 días	lun 12/05/08	vie 08/08/08
11	Puesta en funcionamiento	5 días	lun 11/08/08	vie 15/08/08
12	Pruebas finales	10 días	mié 13/08/08	mar 26/08/08
13	**- Documentación**	**127 días**	**lun 03/03/08**	**mar 26/08/08**
14	Documentación del problema propuesto	5 días	lun 03/03/08	vie 07/03/08
15	Documentación de la investigación	10 días	lun 10/03/08	vie 21/03/08
16	Documentación del análisis	18 días	lun 24/03/08	mié 16/04/08
17	Documentacioón del diseño	21 días	lun 07/04/08	lun 05/05/08
18	Manual técnico y de usuario	81 días	mar 06/05/08	mar 26/08/08

Figura 5. Duración estimada.

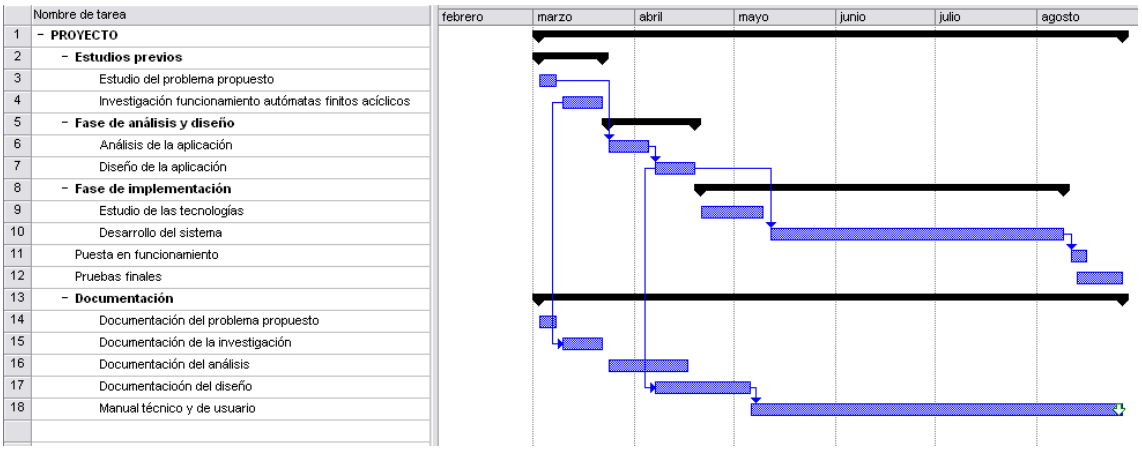

Figura 6. Diagrama de Gantt inicial.

4.2. PLANIFICACIÓN REAL

Se puede observar que las fases de desarrollo reales fueron las mismas que las previstas, sin embargo se produjeron ciertas desviaciones en el tiempo. La planificación inicial del proyecto se calculaba en 25,4 semanas y finalmente ha sido de 37,8. El retraso de la planificación real ha sido, fundamentalmente, debido a la inexperiencia en la realización de planificaciones de este tipo y al optimismo a la hora de realizarla. El estudio e investigación del problema propuesto, el funcionamiento de los autómatas finitos acíclicos, las tecnologías a emplear y el propio desarrollo del sistema supusieron más tiempo del planeado. También algunos contratiempos hicieron que en algunos momentos la jornada de trabajo no fuera la que inicialmente se preveyó.

	Nombre de tarea	Duración	Comienzo	Fin
1	**- PROYECTO**	**189 días**	**lun 03/03/08**	**jue 20/11/08**
2	**- Estudios previos**	**25 días**	**lun 03/03/08**	**vie 04/04/08**
3	Estudio del problema propuesto	10 días	lun 03/03/08	vie 14/03/08
4	Investigación funcionamiento autómatas finitos acíclicos	15 días	lun 17/03/08	vie 04/04/08
5	**- Fase de análisis y diseño**	**36 días**	**lun 07/04/08**	**lun 26/05/08**
6	Análisis de la aplicación	15 días	lun 07/04/08	vie 25/04/08
7	Diseño de la aplicación	21 días	lun 28/04/08	lun 26/05/08
8	**- Fase de implementación**	**105 días**	**mar 27/05/08**	**lun 20/10/08**
9	Estudio de las tecnologías	15 días	mar 27/05/08	lun 16/06/08
10	Desarrollo del sistema	90 días	mar 17/06/08	lun 20/10/08
11	Puesta en funcionamiento	10 días	mar 21/10/08	lun 03/11/08
12	Pruebas finales	15 días	vie 31/10/08	jue 20/11/08
13	**- Documentación**	**189 días**	**lun 03/03/08**	**jue 20/11/08**
14	Documentación del problema propuesto	10 días	lun 03/03/08	vie 14/03/08
15	Documentación de la investigación	15 días	lun 17/03/08	vie 04/04/08
16	Documentación del análisis	15 días	lun 07/04/08	vie 25/04/08
17	Documentacioón del diseño	30 días	lun 28/04/08	vie 06/06/08
18	Manual técnico y de usuario	119 días	lun 09/06/08	jue 20/11/08

Figura 7. Duración real.

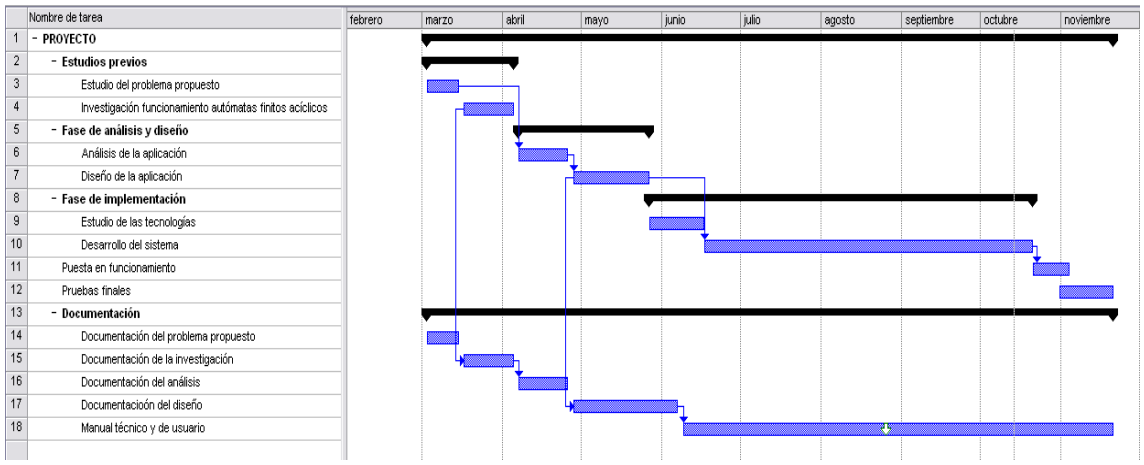

Figura 8. Diagrama de Gantt final

PRESUPUESTO

Coste de desarrollo del proyecto dividido entre los costes de recursos físicos y recursos humanos.

El presupuesto necesario para desarrollar el proyecto se establece mediante el coste de los recursos físicos, tanto a nivel de software como a nivel de hardware y a la cuantía referente a los recursos humanos.

1. RECURSOS FÍSICOS

Los recursos físicos empleados durante el desarrollo del proyecto son los que conforman el puesto de trabajo.

El empleo de software libre no genera costes:

- Sistema operativo GNU/Linux. Distribución Ubuntu 8.10.
- Editor Netbeans versión 6.0.1 para C en Ubuntu con el manejo de las librerías glib 2.12 y libxml2.
- Herramienta para el modelado: Visual paradigm communitiy edition (versión libre disponible para alumnos de la escuela superior de ingeniería informática en el campus de Ourense).

- Paquete ofimático para documentación: Microsoft Word 2007 (versión libre disponible para alumnos de la escuela superior de ingeniería informática en el campus de Ourense).
- Windows XP (versión libre disponible para alumnos de la escuela superior de ingeniería informática en el campus de Ourense), necesario únicamente para manejar el paquete ofimático.
- Planificación temporal: planner.

Recursos Físicos	Descripción	Coste
Puesto de trabajo	Procesador Intel Pentium III a 1000 MHZ o superior Memoria RAM 216 MB Disco Duro 1GB Monitor Teclado Ratón Unidad de CD-ROM Otros (impresora, USB…)	1000 € / 10[1]
Total (I.V.A. incluido)		100 €

2. RECURSOS HUMANOS

El desarrollador del proyecto en este caso ha sido una única persona, pero dependiendo de la tarea realizada genera costes diferentes.

Recursos Humanos	Descripción	Coste Planificado
Analista (40 €/hora)	25 días x 7 horas/día = 175 horas	175 horas x 40 €/hora = 7.000 €
Diseñador (35 €/hora)	10 días x 7 horas/día = 70 horas	70 horas x 35 €/hora = 2.450 €
Programador (30 €/hora)	80 días x 7 horas/día = 560 horas	560 horas x 30 €/hora = 16.800 €
Total (I.V.A. incluido)		26.250 €

Puede observarse que de haber seguido la planificación temporal real el presupuesto variaría.

Recursos Humanos	Descripción	Coste Real
Analista (40 €/hora)	40 días x 5 horas/día = 200 horas	200 horas x 40 €/hora = 8.000 €
Diseñador (35 €/hora)	21 días x 5 horas/día = 105 horas	105 horas x 35 €/hora = 3.675 €
Programador (30 €/hora)	105 días x 5 horas/día = 525 horas	525 horas x 30 €/hora = 15.750 €
Total (I.V.A. incluido)		27.425 €

[1] Tanto hardware como software están amortizados a diez proyectos, como regla general en el desarrollo de aplicaciones.

3. PRESUPUESTO TOTAL

Finalmente se obtiene el importe total del presupuesto que sería algo mayor de haber seguido la planificación real del proyecto.

Coste Total	Coste Total Planificado	Coste Total Real
Recursos Físicos	100 €	100 €
Recursos Humanos	26.250 €	27.425 €
Total (I.V.A. incluido)	26.350 €	27.525 €

CONCLUSIONES

Después de completar el desarrollo del proyecto, se llega a una serie de consideraciones.

Una vez finalizado el desarrollo del proyecto, llega el momento de mirar atrás y reflexionar sobre los resultados obtenidos, lo aprendido, las dificultades encontradas y comprobar si se ha llegado a las metas impuestas en las planificaciones iniciales.

La realización de este proyecto me ha permitido adentrarme en el manejo de herramientas de software libre, hoy en día en pleno auge (y antes prácticamente desconocidas para mí) lo que me ha hecho cambiar mis preferencias. A día de hoy si puedo decidir elijo emplear software libre en detrimento de software propietario, ya que han resultado para mí herramientas de fácil aprendizaje y de manejo intuitivo.

Partiendo desde la planificación inicial que aparece en el anteproyecto se puede observar que está lejos del desarrollo final del proyecto. Esto demuestra la dificultad que supone el realizar planificaciones a priori. La clave para acertar en una planificación de este tipo es la experiencia en haber realizado otros proyectos de tamaño y características similares. Existen mecanismos de estimación de costes que son de gran ayuda para la planificación, pero por desgracia también están basados en el conocimiento de planificaciones y ejecuciones de otros proyectos. Disponer de planificaciones de proyectos similares permite refinar las planificaciones, y no ha sido el caso.

El uso de la notación UML para realizar el análisis y el diseño de la aplicación me ha permitido no sólo perfeccionar mis conocimientos sobre ella, sino también apreciar su importancia, ya que me ha facilitado los pasos a seguir a la hora de realizar la implementación de cada parte. Además, no he encontrado dificultades a la hora de adaptarlo a la arquitectura de programación estructurada seleccionada para la parte de implementación. Referente a la metodología UML, hay que decir que la libertad de trabajo que proporciona esta metodología, provoca a veces indecisiones en la fase de análisis y diseño que fueron solventadas intentando seguir el orden más sencillo posible.

Debido a que C ha sido el lenguaje elegido para realizar este proyecto, he podido adquirir los conocimientos necesarios para desenvolverme satisfactoriamente y corroborar que la elección de este lenguaje es la más acertada para este tipo de aplicaciones donde la rapidez en la ejecución y el ahorro de memoria cobran especial importancia.

No han sido pocos los problemas encontrados, aunque el desarrollo del proyecto ha seguido un avance continuo sin estancamientos acentuados. Las mayores dificultades se han dado en la localización de fallos con el editor de C, casi todos ellos relacionados con la falta de conocimiento en la materia, aunque poco a poco se han podido solucionar las dudas.

A parte de los conocimientos en cuanto a software, he podido aprender todo el funcionamiento interno de autómatas y la fase de análisis léxico y creo que todo ello me permitiría realizar cualquier tipo de aplicación para el manejo de este tipo de estructuras, por supuesto en un tiempo más reducido y con mayor facilidad.

Finalmente concluyo diciendo que personalmente creo que el resultado final ha sido el deseado, ya que cumple los requisitos impuestos inicialmente. El proceso de refinamiento y pruebas ha sido lo más costoso del proyecto pero a la vez el más satisfactorio al ver alcanzados los resultados esperados.

1. AMPLIACIONES

Existen algunas ampliaciones que podrían ser interesantes para nuestro sistema de recuperación de información.

En cuanto a la información asociada a las palabras del diccionario, en el sistema diseñado se organiza en columnas y los tipos de datos empleados para manipular dicha información son enteros y flotantes. Quizás sea útil añadir algún tipo de dato más para alguna aplicación específica, aunque lo cierto es que, al poder referenciar a otros autómatas, pueden gestionarse la mayor parte de las necesidades.

Otra posible ampliación puede ser la de aumentar el tipo de funciones a aplicar para calcular valores a almacenar en los autómatas previamente compilados, limitadas actualmente a las funciones **palabra_a_índice()** e **índice_a_palabra().**

También sería interesante aplicar las estructuras desarrolladas fuera del contexto de los diccionarios como puede ser la creación de índices de texto para gestores de bases de datos que optimicen, de este modo el acceso a la información.

Por último, dado que el proyecto ha sido planteado en términos de software libre, la publicación, difusión, mantenimiento y seguimiento serán también previsiblemente parte del trabajo futuro.

MANUAL TÉCNICO

INTRODUCCIÓN

Visión previa del contenido de los apartados del manual técnico.

1. DOCUMENTACIÓN TÉCNICA

El manual técnico de este proyecto se corresponde con el desarrollo de las fases de análisis, diseño, implementación y pruebas llevadas a cabo para desarrollar el sistema, etapas que constituyen el ciclo de vida de un sistema informático. Para entender esta parte es necesario conocer la metodología UML.

Por otro lado, con este manual se pretende facilitar en la medida de lo posible el mantenimiento posterior del sistema.

2. SISTEMA IMPLEMENTADO

Como se ha comentado en la memoria de la documentación el objetivo de este proyecto es implementar una librería para el manejo eficiente de diccionarios de gran tamaño, entendiendo por diccionario cualquier estructura que permita acceder a cualquier tipo de información asociada a las palabras en ella almacenada. Esta librería está formada por dos partes: una, que constituye el compilador de diccionarios, y la otra, que ofrece las funciones necesarias para gestionar los diccionarios compilados.

Para realizar la implementación de la librería se ha elegido utilizar el lenguaje de programación C, para garantizar su ejecución en cualquier entorno C o C++ y poder tener control absoluto de la gestión de memoria y ejecución (algo imprescindible para este proyecto), la librería glib, para asegurar la portabilidad entre máquinas, y libxml2 para el acceso al archivo XML de configuración del compilador de diccionarios.

Por otra parte, se ha utilizado el paradigma de programación estructurada para organizar adecuadamente los tipos de datos y las funciones. Para el análisis y el diseño se ha empleado UML que, si bien está más pensado para el paradigma de la programación orientada a objetos, se ha convertido paulatinamente en el lenguaje de modelado estándar y puede emplearse perfectamente en el caso de optar por programación estructurada.

La arquitectura sobre la que se ha desarrollado la librería ha sido GNU/Linux, garantizando un entorno robusto, estable y rápido. Esto no descarta que, si glib y libxml2 funcionan correctamente en otras plataformas, la librería tampoco debería tener problemas de funcionamiento en estos entornos.

El almacenamiento de las palabras del diccionario se realiza mediante el empleo de un autómata finito acíclico determinista numerado. Un autómata finito (en adelante AF) o máquina de estado finito es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce. En el comienzo del proceso de reconocimiento de una cadena, el AF se encuentra en el estado inicial y a medida que procesa cada símbolo de la cadena va cambiando de estado de acuerdo a lo determinado por la función de transición. Cuando se ha procesado el último de los símbolos de la cadena de entrada, el autómata se detiene. Si el estado en el que se detuvo es un estado de aceptación o final, entonces la cadena pertenece al lenguaje reconocido por el autómata, en caso contrario, la cadena no pertenece a dicho lenguaje. (10)

Formalmente, un autómata finito determinista (AFD) es similar a un autómata de estados finitos, representado con una 5-tupla (S,Σ,T,s,A) donde:

- S un conjunto finito de estados,
- Σ es un alfabeto finito de símbolos de entrada, es decir, el alfabeto de los caracteres que conforman las palabras,
- T es la función que define las transiciones del autómata: $T : S \times \Sigma \to \mathcal{S}$,
- $s \in S$ es el estado inicial,
- $A \subseteq S$ es un conjunto de estados de aceptación o finales.

Al contrario de la definición de autómata finito, este es un caso particular donde no se permiten transiciones vacías, el dominio de la función T es S (con lo cual no se permiten transiciones desde un estado de un mismo símbolo a varios estados). Por lo tanto, un autómata finito determinista es aquel autómata finito cuyo estado de llegada está unívocamente determinado por el estado inicial y el carácter leído por el autómata.

Como se ha mencionado el autómata empleado además de finito será también acíclico y numerado. Un autómata acíclico es un autómata que no tiene ciclos; esto significa que no hay un camino directo que empiece y termine en el mismo vértice. Los estados del autómata estarán etiquetados por números que los identifican en orden ascendente, por eso se dice autómata numerado.

Los autómatas finitos se han venido utilizando frecuentemente para implementar escáneres eficientes (11), y en particular los que reconocen un conjunto finito de palabras utilizados a modo de diccionario para tareas de reconocimiento, corrección de errores e incluso etiquetación.

Existen multitud de maneras de construir este tipo de autómatas que reconocen un conjunto finito de palabras (12), pero muchas de ellas comparten una característica común: consiguen representar las palabras que reconocen de manera muy compacta, lo que hace factible la colocación de grandes diccionarios totalmente en memoria, evitando así el acceso a memoria secundaria para las tareas que requieren el acceso a dicho autómata.

2.1 Compilador

Con respecto a la primera parte de la librería, el compilador parte de un fichero que contiene un listado de las palabras del diccionario junto con su información asociada organizada en columnas, sigue el siguiente formato:

palabra_1 tabulador **información_1** tabulador **información_2** tabulador **información_3** …
palabra_2 tabulador **información_1** tabulador **información_2** tabulador **información_3** …
…

La idea general de su funcionamiento consiste en que el sistema recorre el diccionario fila a fila e inserta cada palabra en el autómata finito siguiendo el algoritmo de Daciuk. Se da la particularidad de que en estos autómatas (numerados) cada palabra tiene asignado un índice que la identifica unívocamente.

Con estos índices se accede a dos tableros de conversión (*mappings*) que contienen las posiciones primera y última respectivamente en los ficheros que almacenan la información (*info*) asociada a las palabras.

Por lo tanto, a medida que se van insertando palabras, todas las estructuras (el propio autómata, los *mappings* y los ficheros con la información) se van modificando hasta alcanzar su configuración final, una vez procesadas todas las palabras.

Finalmente, el compilador escribe todo en un fichero binario de manera comprimida. Este fichero es lo que denominamos diccionario compilado. En la figura 9 se muestra una idea general de las diferentes estructuras que utiliza internamente el compilador.

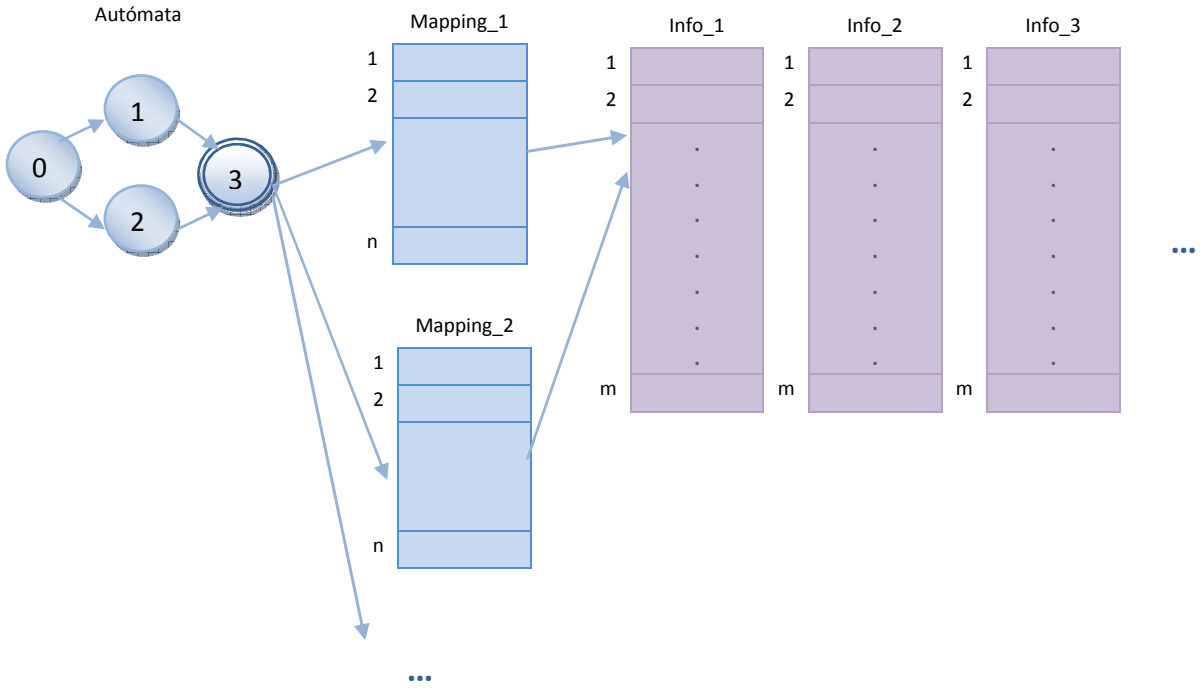

Figura 9. Estructuras utilizadas por el compilador.

2.2 Librería de acceso

Una vez se tiene el/los diccionario/s compilado/s, es la librería de acceso la que proporciona los métodos necesarios para su manejo.

Dicha librería carga en memoria la versión compilada del diccionario y accede al mismo utilizando los métodos que ofrece: cargar un diccionario, obtener la información de una palabra, etc.

La estructura interna que maneja es conceptualmente similar a la presentada en la figura 9 para el compilador, con la salvedad de que la información de las palabras ya no estará en un fichero en disco si no en memoria, y que las estructuras utilizadas para implementar tanto el autómata, como los tableros de conversión y la información, están muy optimizados para el acceso rápido y el escaso consumo de memoria. De hecho, estas estructuras son exactamente las mismas que las utilizadas por el compilador para almacenar el diccionario, es decir, el compilador ya genera el diccionario compilado en disco utilizando estas estructuras óptimas.

3. DICCIONARIO DE EQUIVALENCIAS

Como se ha mencionado en la descripción de la arquitectura, la implementación del sistema sigue un modelo estructurado de programación. El análisis y diseño se ha organizado empleando las siguientes equivalencias entre lo que son los objetos en UML y la materialización en programación estructurada:

Compilador: en la implementación se corresponde con el programa nadfac.c responsable del proceso de compilación del diccionario de palabras.

NADFA: en la implementación es una estructura compuesta a su vez por tres estructuras:

Autómata, Mappings, Infos: en la implementación son estructuras[2] que permiten almacenar los datos del diccionario en el proceso de compilación.

estructura NADFA {

estructura NADFA_Automata;

estructura NADFA_Mappings;

estructura NADFA_Infos;

};

[2] Una estructura es un grupo de variables las cuales pueden ser de diferentes tipos sostenidas o mantenidas juntas en una sola unidad. La unidad es la estructura

estructura NADFA_Automata {

array[3] de estados;

array de transiciones del autómata;

};

estructura NADFA_Mapping {

array de índices de las palabras;

};

estructura NADFA_Info {

array de columnas de información;

};

NADFA_BIN: en la implementación es una estructura que almacena en memoria las palabras del diccionario comprimido recuperado del fichero binario, el índice de la información y dicha información, que interviene en la implementación de las funciones de la librería en libnadfa_bin.c.

estructura NADFA_BIN {

array de celdas que componen el autómata;

array de índices de las palabras del autómata;

arrays de datos de las informaciones asociadas a las palabras;

};

E/S: representa el acceso a la entrada/salida del sistema.

Según las equivalencias presentadas, en resumen lo que en la implementación es una estructura, en análisis y diseño funciona como un objeto en orientación a objetos.

[3] En programación, un vector, matriz, array, arreglo o alineación es un conjunto o agrupación de variables del mismo tipo cuyo acceso se realiza por índices. Desde el punto de vista de un programa de ordenador, un array (matriz o vector) es una zona de almacenamiento contiguo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico un array se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).

ANÁLISIS

La fase de análisis define el sistema intentando responder a la pregunta ¿Qué es lo que el sistema debe hacer?

1. INTRODUCCIÓN

A continuación se describe la etapa de análisis del sistema desarrollado. En esta etapa se analizan los requisitos del proyecto, estudiando la interacción de los usuarios con el sistema para saber, una vez finalizada esta etapa, lo que el sistema debe hacer para satisfacer las necesidades del sistema.

2. DESCRIPCIÓN

En los puntos siguientes se expone cada uno de los diagramas UML utilizados para el desarrollo de la librería, explicándose en cada caso los pormenores del mismo.

Como ya se ha comentado, la librería tiene dos partes bien diferenciadas. Por un lado un compilador que, dada una lista de palabras con su respectiva información y, opcionalmente, algún diccionario ya compilado que se necesite para generar el actual, genera en disco la versión binaria compilada del mismo y, por el otro, la librería propiamente dicha que se encarga del uso y manejo de la versión compilada de los diccionarios.

Por ello, en vez de tomar el sistema como un todo, se ha dividido en dos módulos bien diferenciados:

- **Compilación diccionario:** módulo que abarca todo el proceso de compilación.
- **Acceso diccionario compilado:** módulo compuesto por todas las funciones relativas al manejo de los diccionarios compilados.

3. ANÁLISIS DE REQUISITOS

En este apartado se detallan las especificaciones de los requisitos que describen las funciones que realiza el sistema en su totalidad. La obtención de estos requerimientos se basa en la necesidad del desarrollo de implementaciones de diccionarios que garanticen un rendimiento óptimo en el acceso a las palabras para que no penalicen el tiempo necesario para resolver tareas.

Para poder alcanzar la finalidad de este proyecto y poder así desarrollar una librería que permita la gestión de diccionarios de grandes dimensiones se emplean autómatas finitos acíclicos deterministas numerados, con la finalidad de:

- Minimizar el consumo de memoria necesario para los diccionarios.
- Minimizar el tiempo de ejecución para que puedan ser utilizados en sistemas que necesiten un rendimiento elevado.

La librería está formada por dos partes, una utilizada para construir el compilador encargado de comprimir los diccionarios de palabras “Compilación diccionario”, y la otra responsable de facilitar el acceso y el uso de los diccionarios ya compilados “Gestión diccionario compilado”. Lo que se necesita para compilar los diccionarios (primera parte) es una lista de palabras junto con la información asociada a ellas, y el resultado es un diccionario compilado (comprimido) en formato binario. Después es la segunda parte la que permite gestionar el acceso a los diccionarios compilados (la librería propiamente dicha).

Entre las funcionalidades que tiene esta librería frente a otros trabajos e implementaciones ya existentes destacan:

- Posibilidad de incluir cualquier tipo de información asociada a las palabras.
- Capacidad para gestionar diversos diccionarios simultáneamente y con referencias entre ellos.

De este modo el usuario final puede acceder a las siguientes operaciones:

- Conversión palabra a índice: de una palabra al índice que la representa unívocamente.
- Conversión índice a palabra: de un índice a la palabra que representa.
- Acceso información completa de palabra: acceso a todas las informaciones disponibles de esa palabra en el diccionario (todos los valores de los diferentes campos de información).
- Acceso información concreta de palabra: acceso a la información solicitada de la palabra en el diccionario.
- Índice primera información de una palabra: obtiene el índice en el que aparece la primera información de una palabra.
- Índice última información de una palabra: obtiene el índice en el que aparece la última información de una palabra.
- Cargar diccionario en memoria: carga en memoria el fichero binario que contiene el diccionario compilado.
- Liberar recursos: libera la memoria ocupada por el diccionario compilado.

4. CASOS DE USO

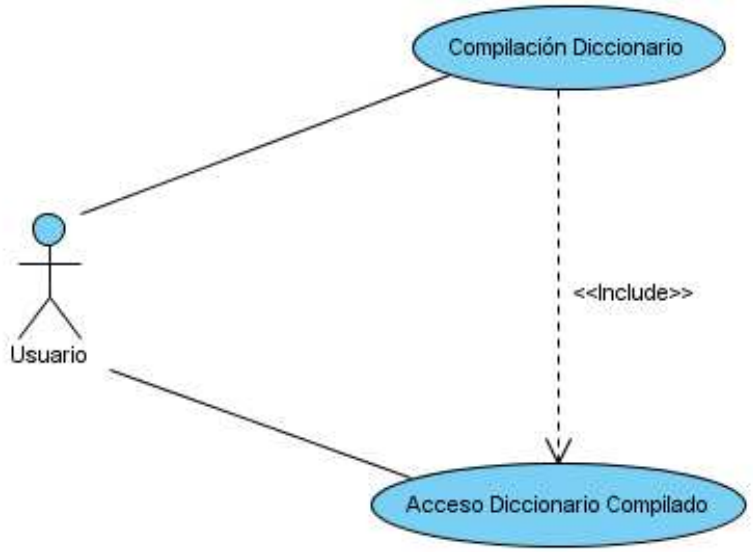

Figura 10. Diagrama de Casos de uso del sistema

4.1. CASO DE USO: COMPILACIÓN DICCIONARIO

COMPILACIÓN DICCIONARIO	
Nombre	Compilación diccionario.
Descripción	Este escenario describe el proceso de compilación que, dada una lista de palabras con su respectiva información y, opcionalmente, algún diccionario ya compilado que se necesite para generar el actual, genera en disco la versión binaria compilada del mismo.
Precondiciones	Que exista el documento inicial con la lista de palabras del diccionario y la información de las mismas. Que exista un fichero de configuración con los parámetros necesarios. Que estén disponibles los diccionarios previamente compilados que puedan ser solicitados.
Flujo Normal	El usuario solicita realizar la compilación del diccionario. El sistema accede al fichero de configuración y carga en memoria los parámetros del mismo.

	El sistema crea en memoria la estructura necesaria para almacenar el autómata que contendrá las palabras del diccionario junto con toda la información necesaria para poder llevar a cabo la compilación. El sistema inserta en el autómata las palabras del diccionario, crea los índices necesarios para acceder a la información de cada palabra, organiza dicha información y la almacena en la estructura anterior. El sistema escribe en disco el autómata junto con el *mapping* y la información ordenadamente, obteniendo así el diccionario compilado.
Flujo Excepcional	Si surge un error con alguno el sistema informa y aborta la operación.
Postcondiciones	El diccionario de palabras está compilado en un fichero binario.

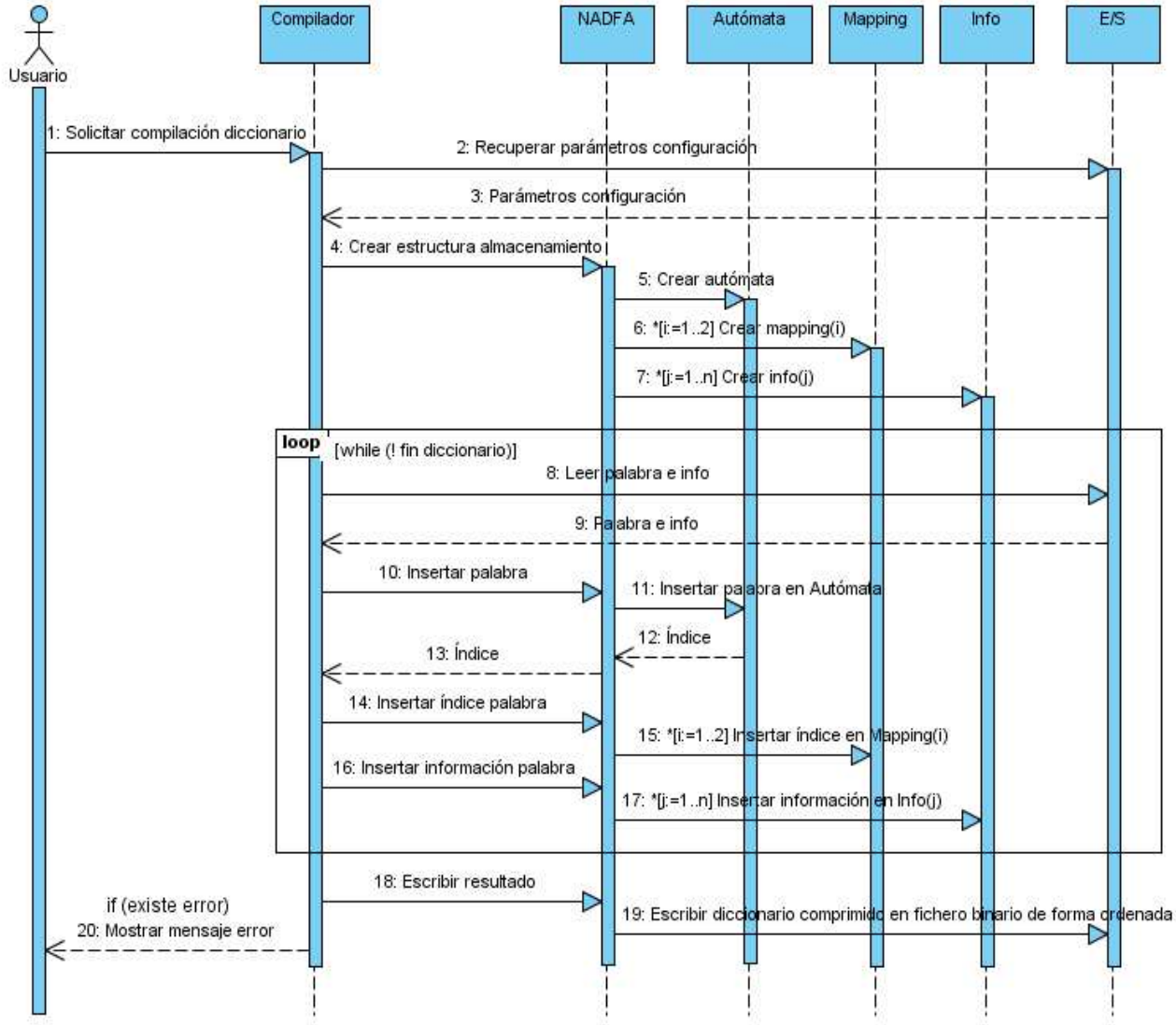

Figura 11. Diagrama de secuencia del caso de uso Compilación Diccionario (Análisis).

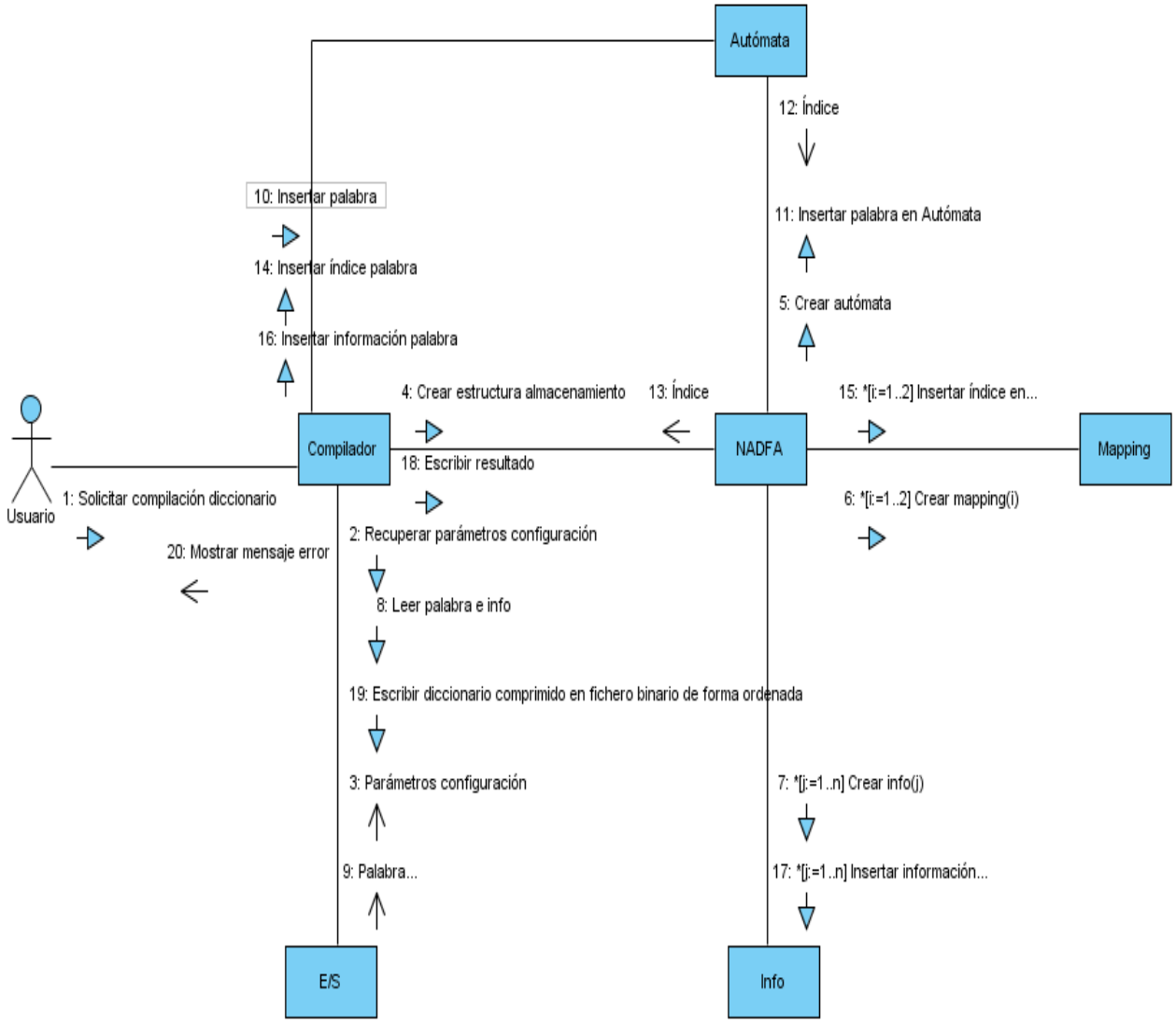

Figura 12. Diagrama de colaboración del caso de uso Compilación Diccionario (Análisis).

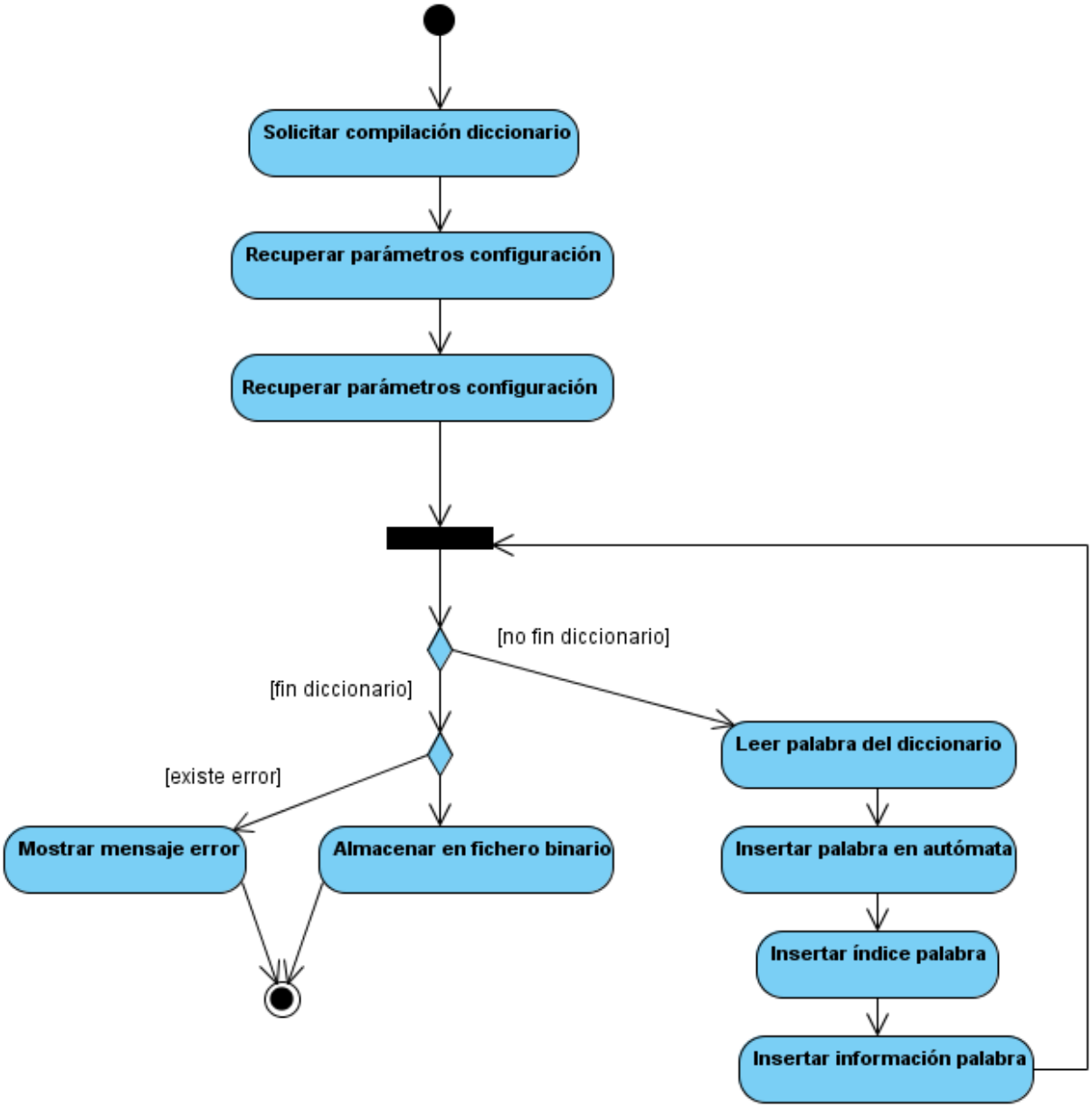

Figura 13. Diagrama de actividades del caso de uso Compilación Diccionario (Análisis).

4.2. CASO DE USO: GESTIÓN DICCIONARIO COMPILADO

Dado que este segundo módulo es una librería, el actor que interactúa con la misma es un posible programa desarrollado, y no el usuario propiamente dicho.

4.2.1 ESCENARIO: CARGAR DICCIONARIO COMPILADO EN MEMORIA

El sistema accede y carga en memoria el fichero binario que contiene el diccionario compilado.

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Cargar diccionario compilado en memoria.
Descripción	Este escenario describe el proceso mediante el que se carga en memoria el fichero binario que contiene el diccionario compilado.
Precondiciones	Que esté disponible el fichero binario que contiene el diccionario de palabras compilado.
Flujo Normal	El programa solicita cargar el diccionario compilado en memoria. El sistema recupera el diccionario compilado del fichero binario. El sistema carga en memoria los datos correspondientes al autómata con las palabras del diccionario. El sistema carga en memoria los datos correspondientes al *mapping*. El sistema carga en memoria los datos correspondientes a la información asociada a las palabras del diccionario.
Postcondiciones	El diccionario compilado está cargado en memoria.

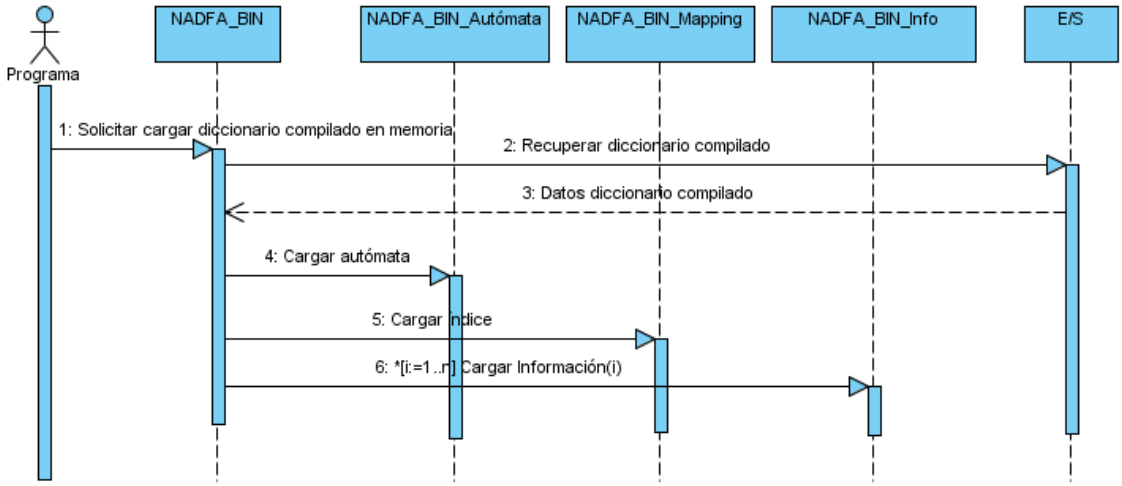

Figura 14. Diagrama de secuencia del caso de uso Cargar diccionario en memoria (Análisis).

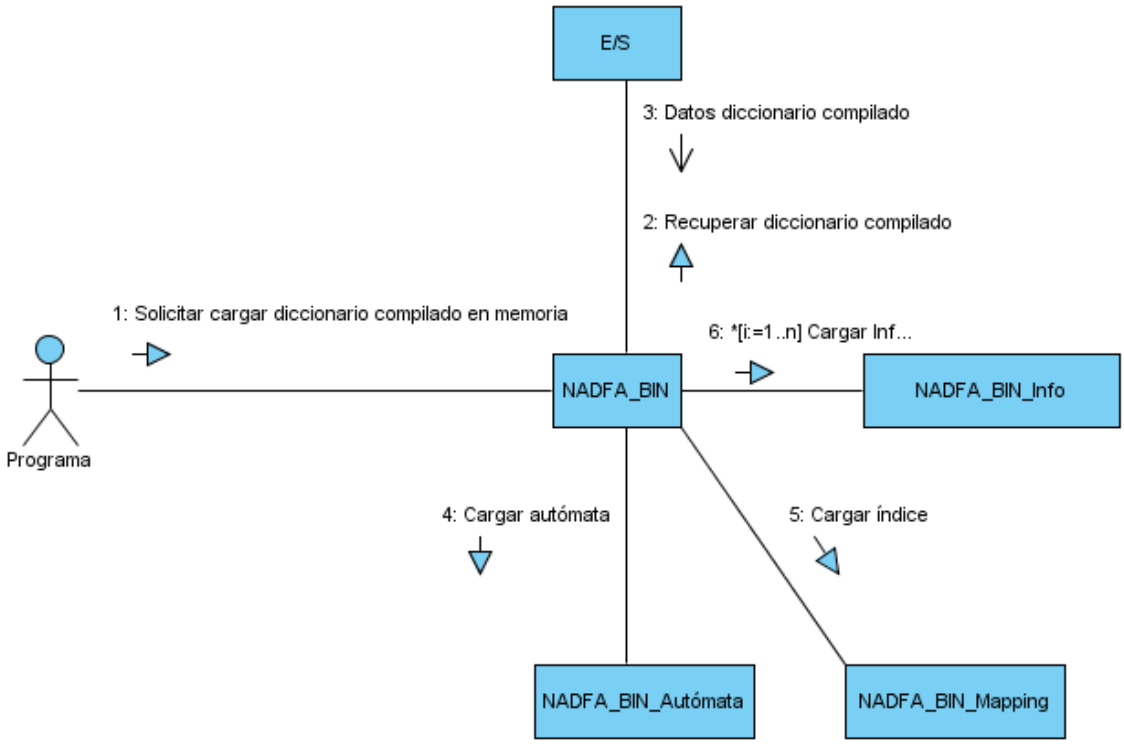

Figura 15. Diagrama de colaboración del caso de uso Cargar diccionario en memoria (Análisis).

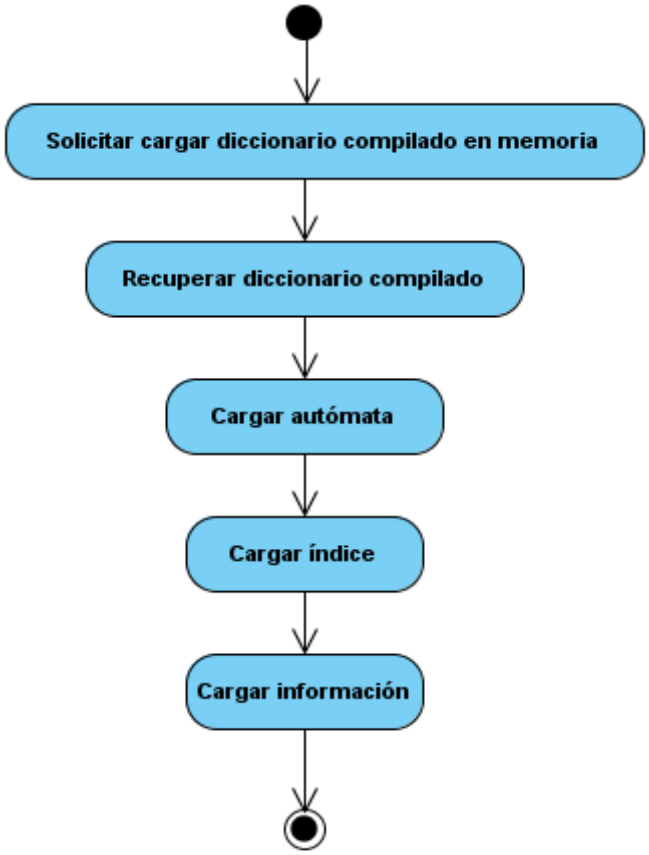

Figura 16. Diagrama de actividades del caso de uso Cargar diccionario compilado en memoria (Análisis).

4.2.2 ESCENARIO: CONVERSIÓN PALABRA A ÍNDICE

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Conversión palabra a índice.
Descripción	Este escenario describe el proceso mediante el que se obtiene el índice que en el autómata compilado se corresponde con la palabra pasada como parámetro.
Precondiciones	Que esté cargado en memoria el fichero binario que contiene el diccionario de palabras compilado.
Flujo Normal	El programa solicita obtener el índice correspondiente a una palabra concreta del diccionario. El sistema accede a la estructura que almacena el autómata con las palabras del diccionario.

	El sistema recorre los estados del autómata transitando por las transiciones de cada estado de manera que devuelve el índice de dicha palabra en el autómata.
Postcondiciones	Se obtiene el índice correspondiente a la palabra dada, cero si la palabra no está en el diccionario.

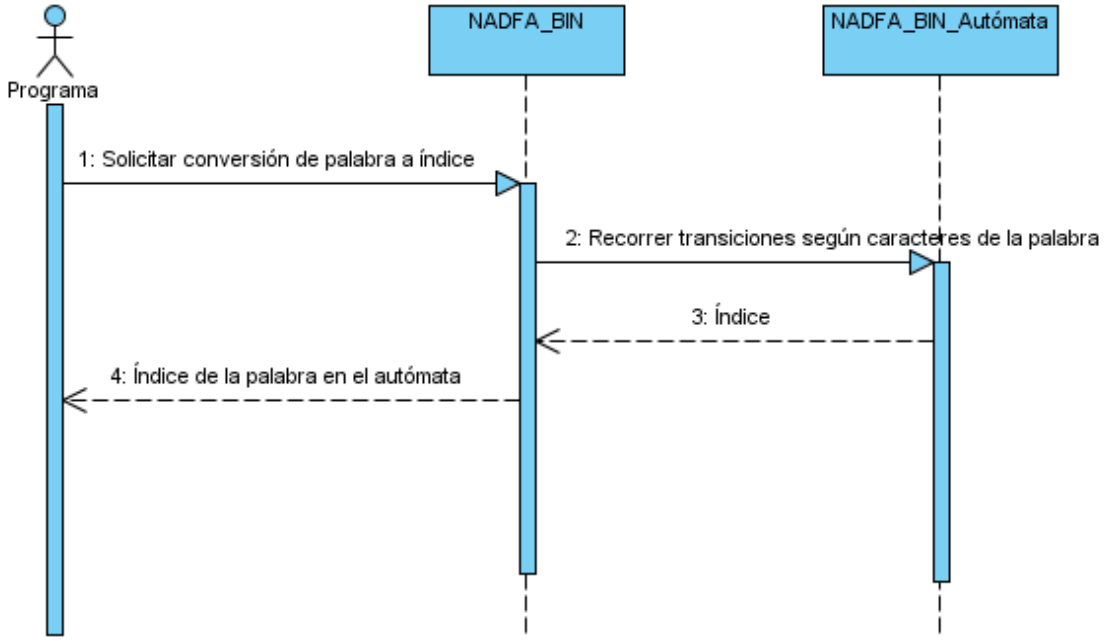

Figura 17. Diagrama de secuencia del caso de uso Conversión Palabra a Índice (Análisis).

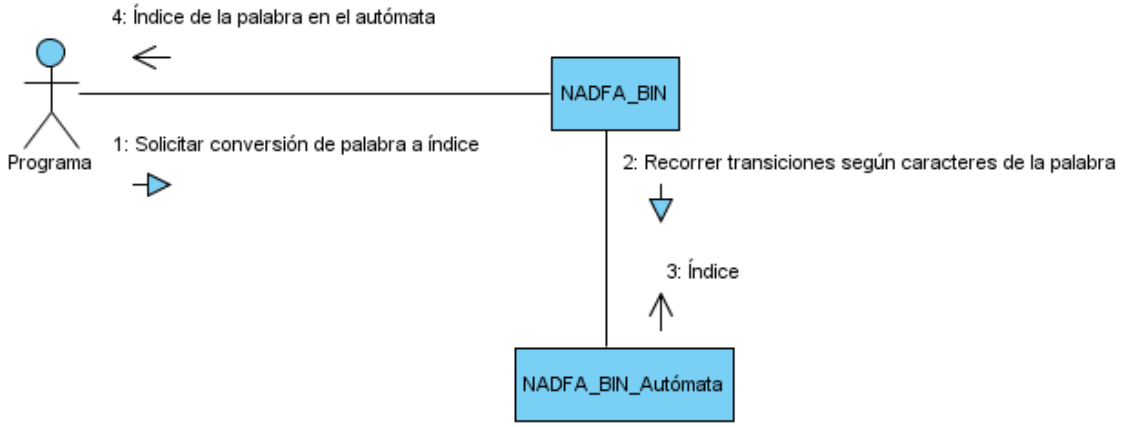

Figura 18. Diagrama de colaboración del caso de uso Conversión Palabra a Índice (Análisis).

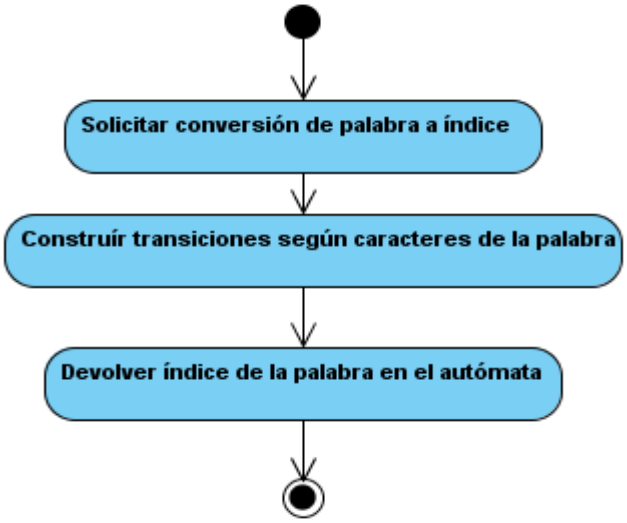

Figura 19. Diagrama de actividades del caso de uso Conversión Palabra a Índice (Análisis)

4.2.3 ESCENARIO: CONVERSIÓN ÍNDICE A PALABRA

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Conversión índice a palabra.
Descripción	Este escenario describe el proceso mediante el cual se obtiene una palabra del autómata compilado a partir del índice, o identificador unívoco, que se pasa como parámetro.
Precondiciones	Que esté cargado en memoria el fichero binario que contiene el diccionario de palabras compilado.
Flujo Normal	El programa solicita obtener la palabra correspondiente a un índice concreto del autómata de palabras. El sistema accede a la estructura que almacena el autómata con las palabras del diccionario. El sistema recorre los estados del autómata hasta encontrar el índice. El sistema construye la palabra correspondiente a dicho índice transitando letra a letra por los estados del autómata.
Postcondiciones	Se obtiene la palabra correspondiente al índice dado, la palabra vacía si el índice no se corresponde con ninguna palabra del diccionario.

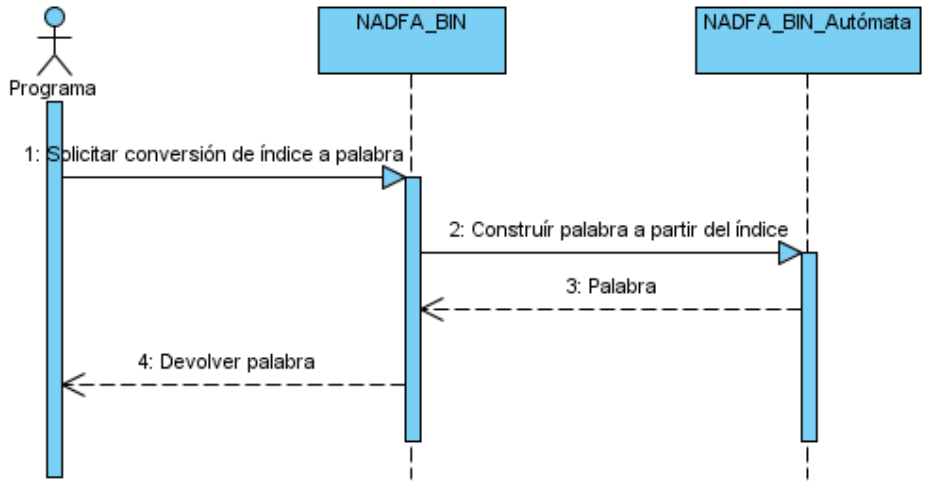

Figura 20. Diagrama de secuencia del caso de uso Conversión Índice a Palabra (Análisis).

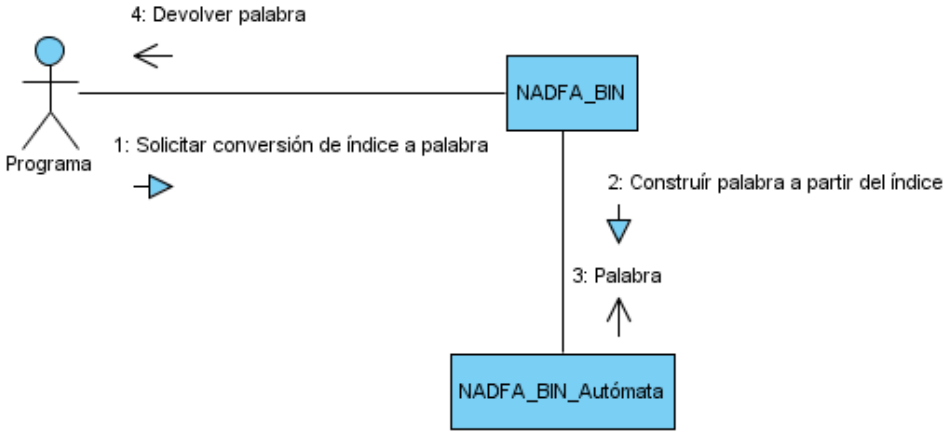

Figura 21. Diagrama de colaboración del caso de uso Conversión Índice a Palabra (Análisis).

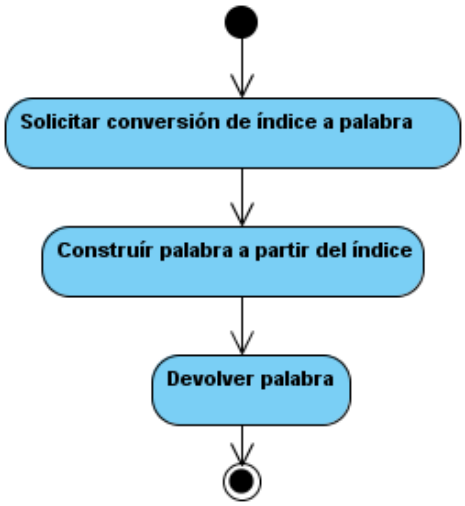

Figura 22. Diagrama de actividades del caso de uso Conversión índice a Palabra (Análisis).

4.2.4 ESCENARIO: ACCEDER INFORMACIÓN COMPLETA DE PALABRA

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Acceder información completa de palabra.
Descripción	Este escenario describe el proceso que permite obtener toda la información almacenada de una palabra del diccionario.
Precondiciones	Que esté cargado en memoria el fichero binario que contiene el diccionario de palabras compilado. Que estén disponibles los diccionarios previamente compilados que puedan ser solicitados.
Flujo Normal	El programa solicita obtener la información de una palabra del diccionario que debe introducir como parámetro. El sistema accede a la estructura que almacena el autómata con las palabras del diccionario. El sistema busca el índice de la primera información de esa palabra. El sistema calcula el número total de entradas de información que de esa palabra hay en el diccionario compilado. El sistema devuelve toda la información de todas las entradas que hay en el diccionario compilado para esa palabra.
Postcondiciones	Se obtiene toda la información de una palabra determinada, la cadena vacía si la palabra no está en el diccionario.

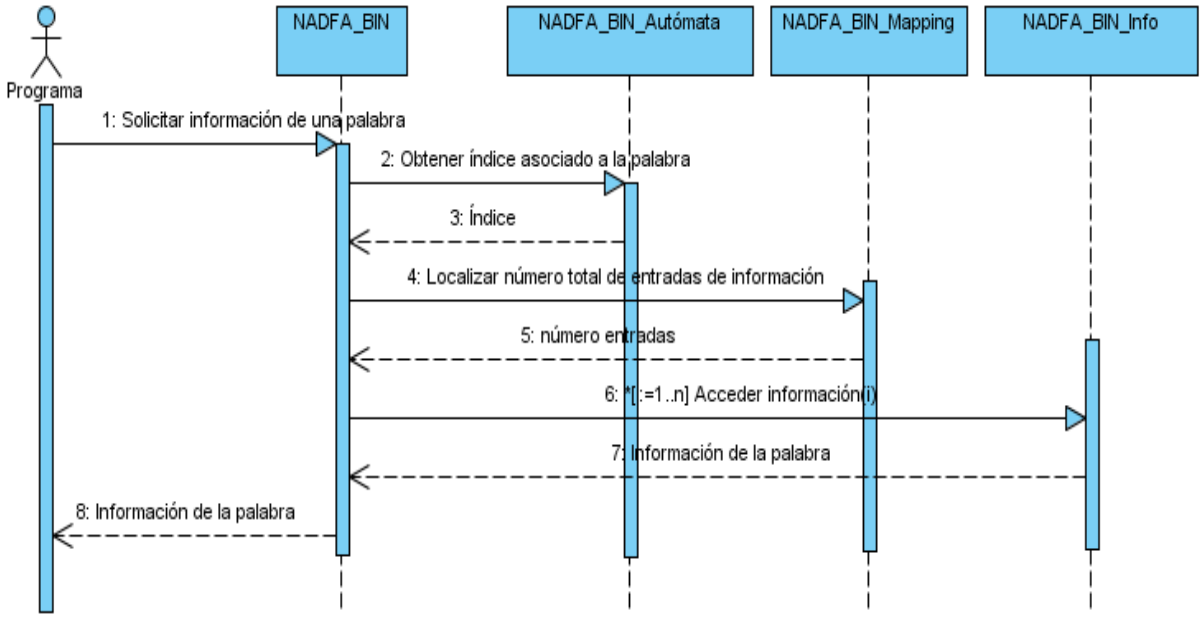

Figura 23. Diagrama de secuencia del caso de uso Acceder Información Completa de Palabra (Análisis).

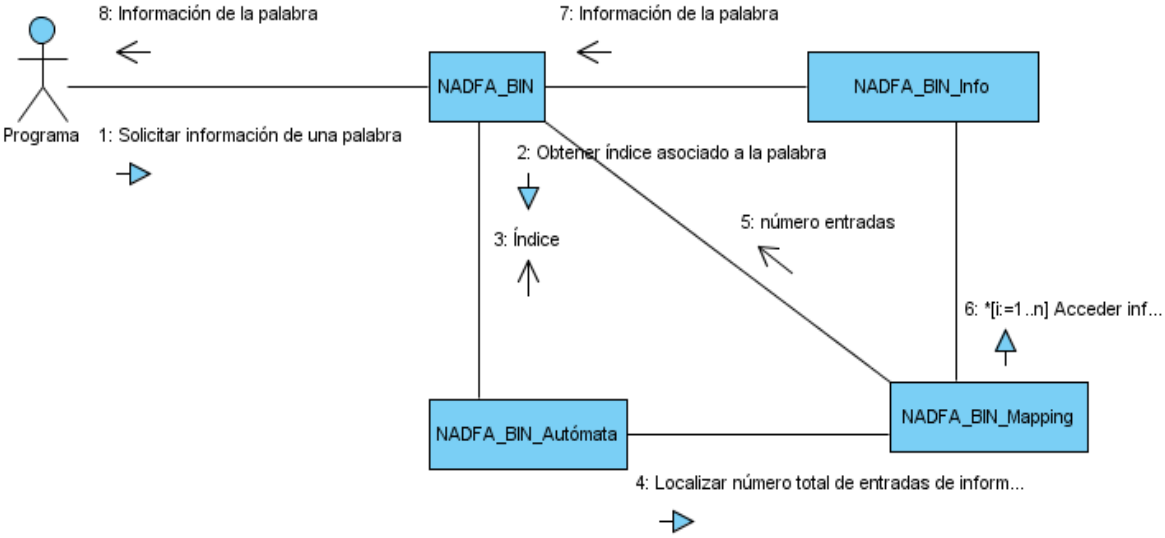

Figura 24. Diagrama de colaboración del caso de uso Acceder Información Completa de Palabra (Análisis).

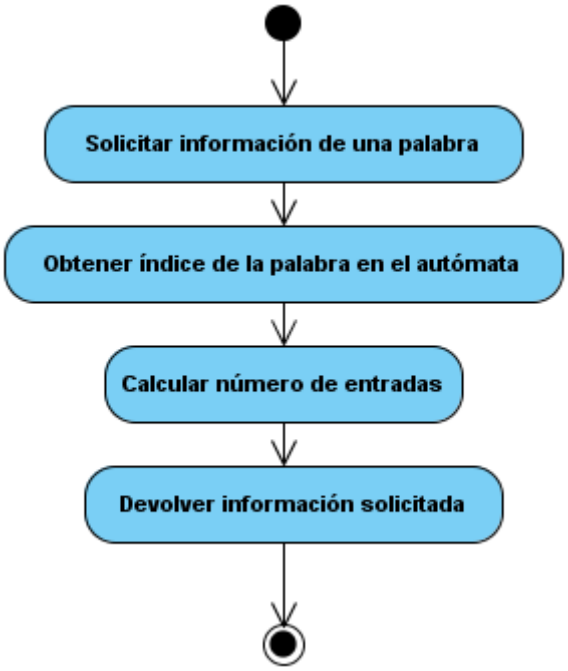

Figura 25. Diagrama de actividades del caso de uso Acceder Información Completa de Palabra (Análisis).

4.2.5 ESCENARIO: ACCEDER INFORMACIÓN CONCRETA DE PALABRA

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Acceder información concreta de palabra.
Descripción	Este escenario describe el proceso que permite obtener una información concreta de una palabra del diccionario.
Precondiciones	Que esté cargado en memoria el fichero binario que contiene el diccionario de palabras compilado. Que estén disponibles los diccionarios previamente compilados que puedan ser solicitados.
Flujo Normal	El programa solicita obtener un tipo de información de una palabra del diccionario que debe introducir como parámetro. El sistema accede a la estructura que almacena el autómata con las palabras del diccionario. El sistema busca el índice asociado a esa palabra en el autómata.

	El sistema calcula el número total de entradas de información que de esa palabra hay en el diccionario compilado. El sistema obtiene la información solicitada de todas las entradas que hay en el diccionario compilado para esa palabra.
Postcondiciones	Se obtiene la información solicitada de una palabra determinada, la cadena vacía si la palabra no está en el diccionario.

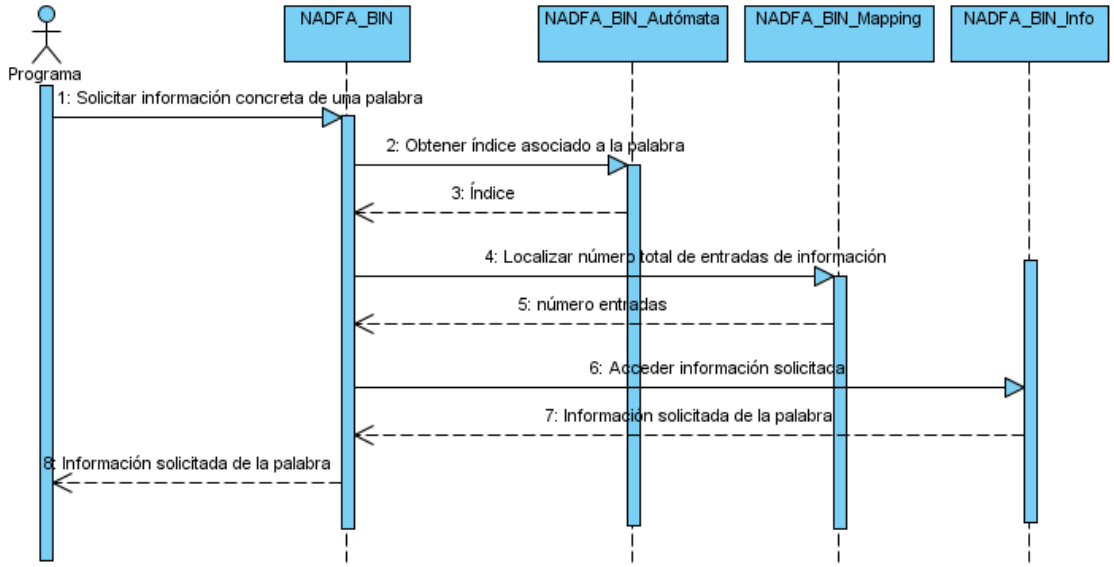

Figura 26. Diagrama de secuencia del caso de uso Acceder Información Concreta de Palabra (Análisis).

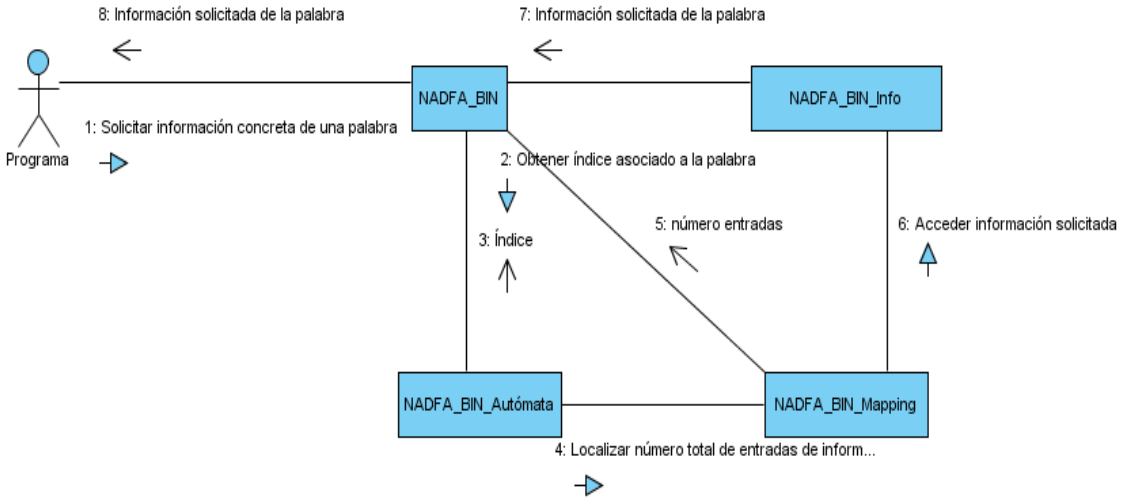

Figura 27. Diagrama de colaboración del caso de uso Acceder Información Concreta de Palabra (Análisis).

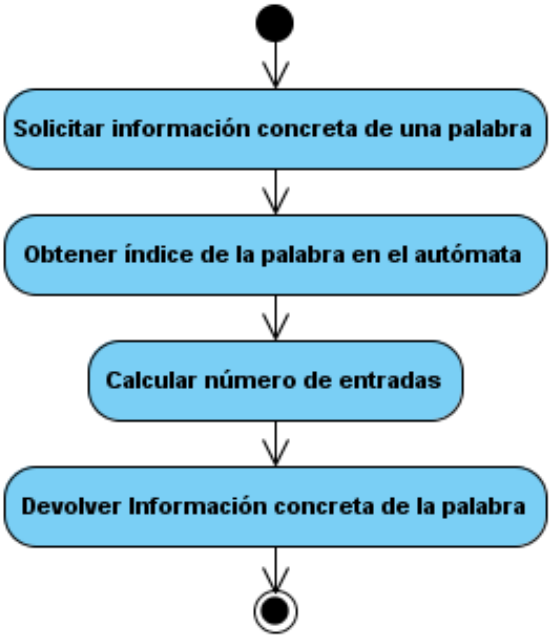

Figura 28. Diagrama de actividades del caso de uso Acceder Información Concreta de Palabra (Análisis).

4.2.6 ESCENARIO: ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Índice primera información de una palabra.
Descripción	Este escenario describe el proceso que permite obtener la posición de la primera información de una palabra en el diccionario.
Precondiciones	Que esté disponible el fichero binario que contiene el diccionario de palabras compilado. Que estén disponibles los diccionarios previamente compilados que puedan ser solicitados.
Flujo Normal	El programa solicita obtener el índice de la primera información de una palabra. El sistema accede a la estructura que almacena el autómata con las palabras del diccionario. El sistema obtiene el índice de la primera información de esa palabra en el autómata.

Postcondiciones	Se obtiene el índice de la primera información de la palabra en el diccionario compilado, cero si la palabra no está en el diccionario.

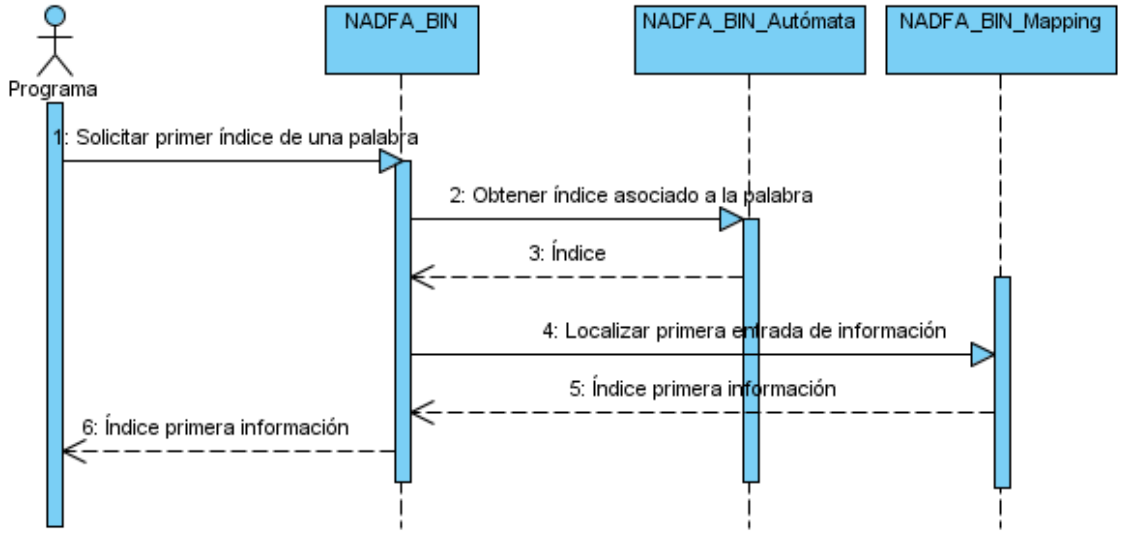

Figura 29. Diagrama de secuencia del caso de uso Índice Primera Información de una Palabra (Análisis).

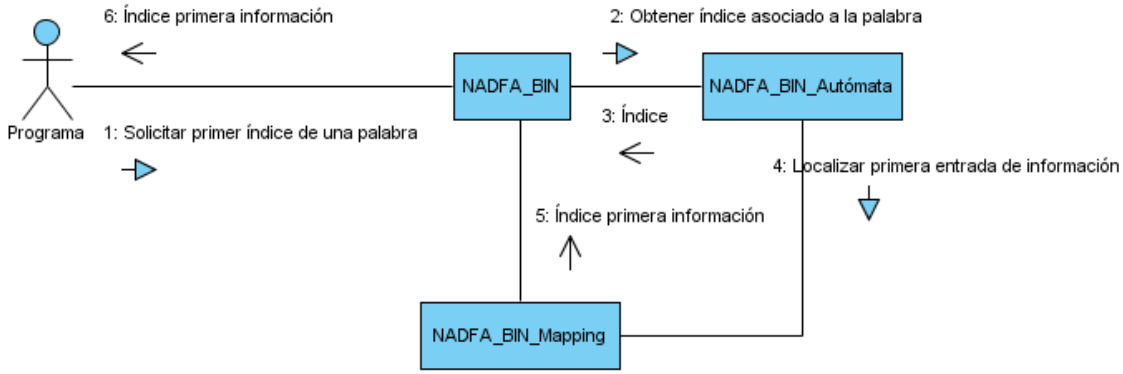

Figura 30. Diagrama de colaboración del caso de uso Índice Primera Información de una Palabra (Análisis).

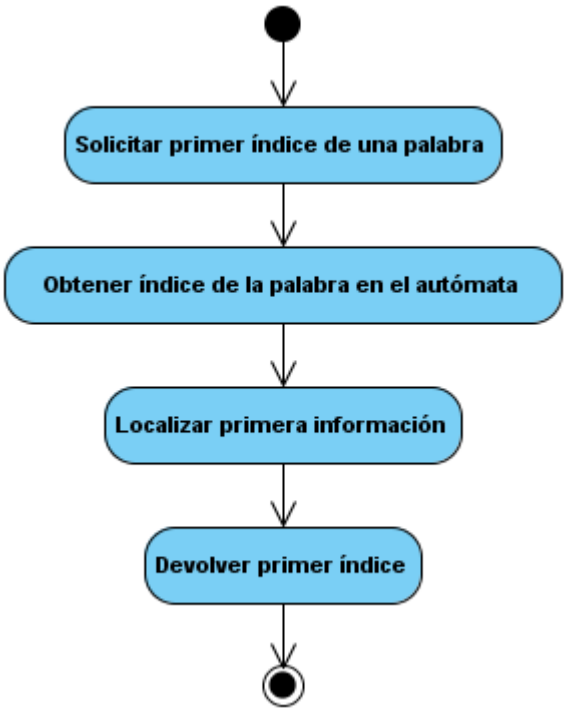

Figura 31. Diagrama de actividades del caso de uso Índice Primera Información de una Palabra (Análisis).

4.2.7 ESCENARIO: ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Índice última información de una palabra.
Descripción	Este escenario describe el proceso que permite obtener la posición de la última información de una palabra en el diccionario.
Precondiciones	Que esté cargado en memoria el fichero binario que contiene el diccionario de palabras compilado. Que estén disponibles los diccionarios previamente compilados que puedan ser solicitados.
Flujo Normal	El programa solicita obtener el índice de la última información de una palabra. El sistema accede a la estructura que almacena el autómata con las palabras del diccionario. El sistema busca el índice de la primera aparición de la siguiente

	palabra a la solicitada y devuelve la posición anterior, que es la equivalente a la última información de la palabra solicitada.
Postcondiciones	Se obtiene el índice de la última información de la palabra en el diccionario compilado, cero si la palabra no está en el diccionario.

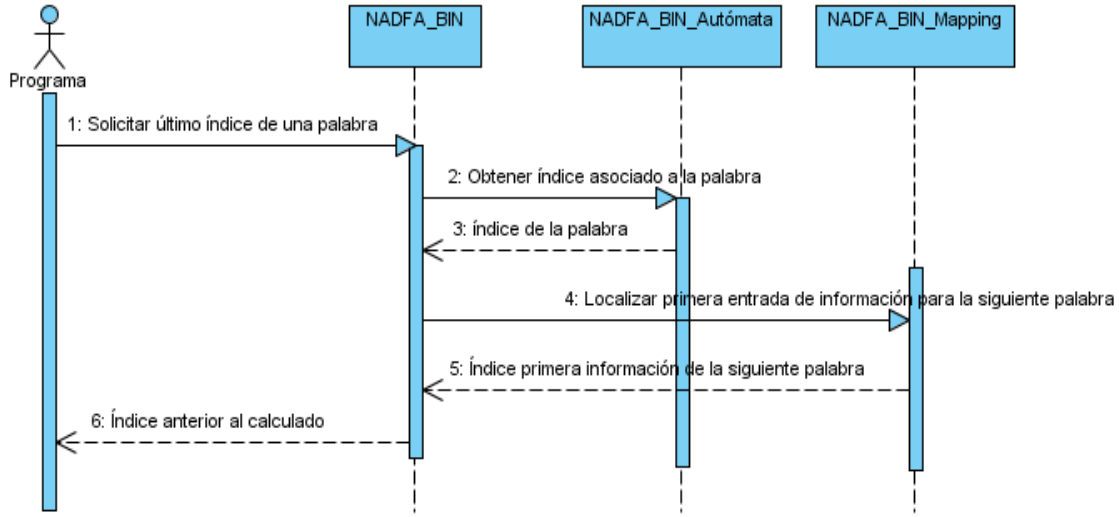

Figura 32. Diagrama de secuencia del caso de uso Índice Última Información de una Palabra (Análisis).

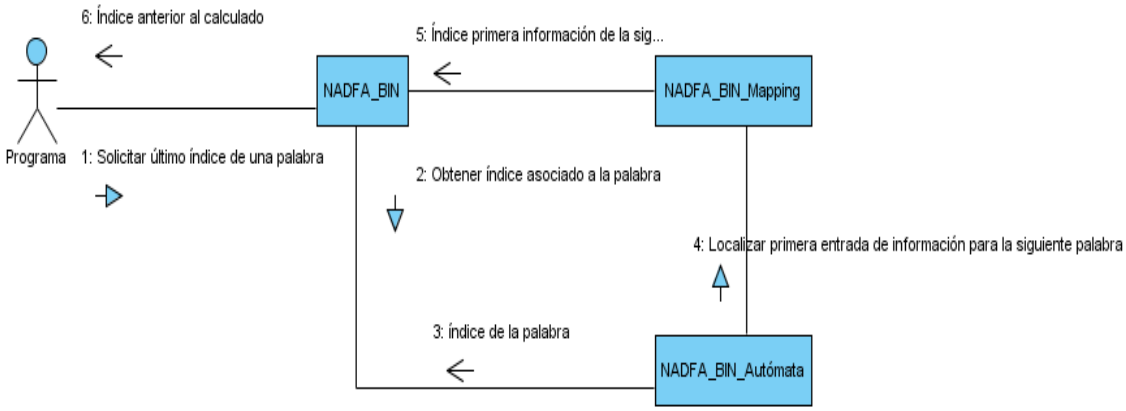

Figura 33. Diagrama de colaboración del caso de uso Índice Última Información de una Palabra (Análisis).

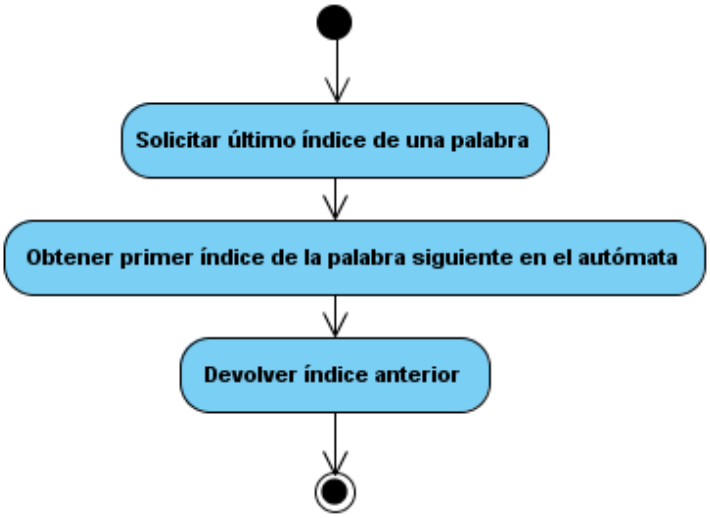

Figura 34. Diagrama de actividades del caso de uso Índice Última Información de una Palabra (Análisis).

4.2.8 ESCENARIO: LIBERAR RECURSOS

GESTIÓN DICCIONARIO COMPILADO	
Nombre	Liberar recursos.
Descripción	Este escenario describe el proceso que permite liberar la memoria ocupada para el almacenamiento del diccionario compilado.
Precondiciones	Que esté cargado en memoria el fichero binario que contiene el diccionario de palabras compilado.
Flujo Normal	El programa solicita liberar los recursos de memoria. El sistema accede a la memoria que ocupa el autómata con las palabras del diccionario y la libera. El sistema accede a la memoria que ocupa el *mapping* y la libera. El sistema accede a la memoria que ocupa la información del diccionario y la libera. El sistema libera la memoria de la estructura que agrupa todos los elementos del diccionario.
Postcondiciones	No hay memoria ocupada por el diccionario compilado.

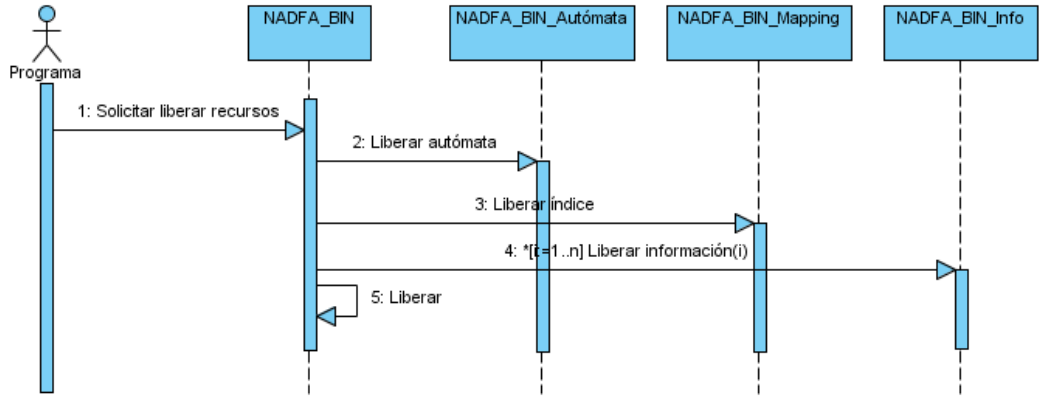

Figura 35. Diagrama de secuencia del caso de uso Liberar recursos (Análisis).

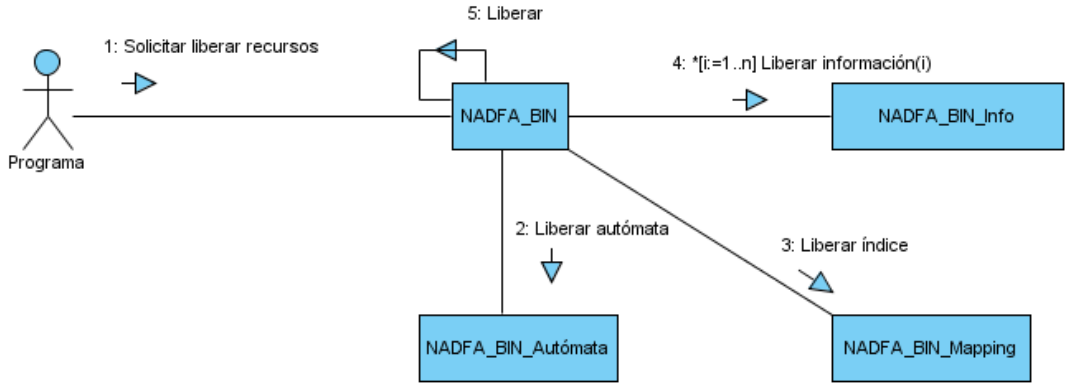

Figura 36. Diagrama de colaboración del caso de uso Liberar recursos (Análisis).

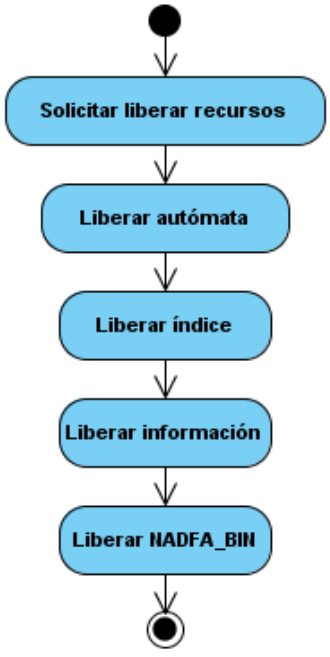

Figura 37. Diagrama de actividades del caso de uso Liberar recursos (Análisis).

5. DIAGRAMA DE CLASES

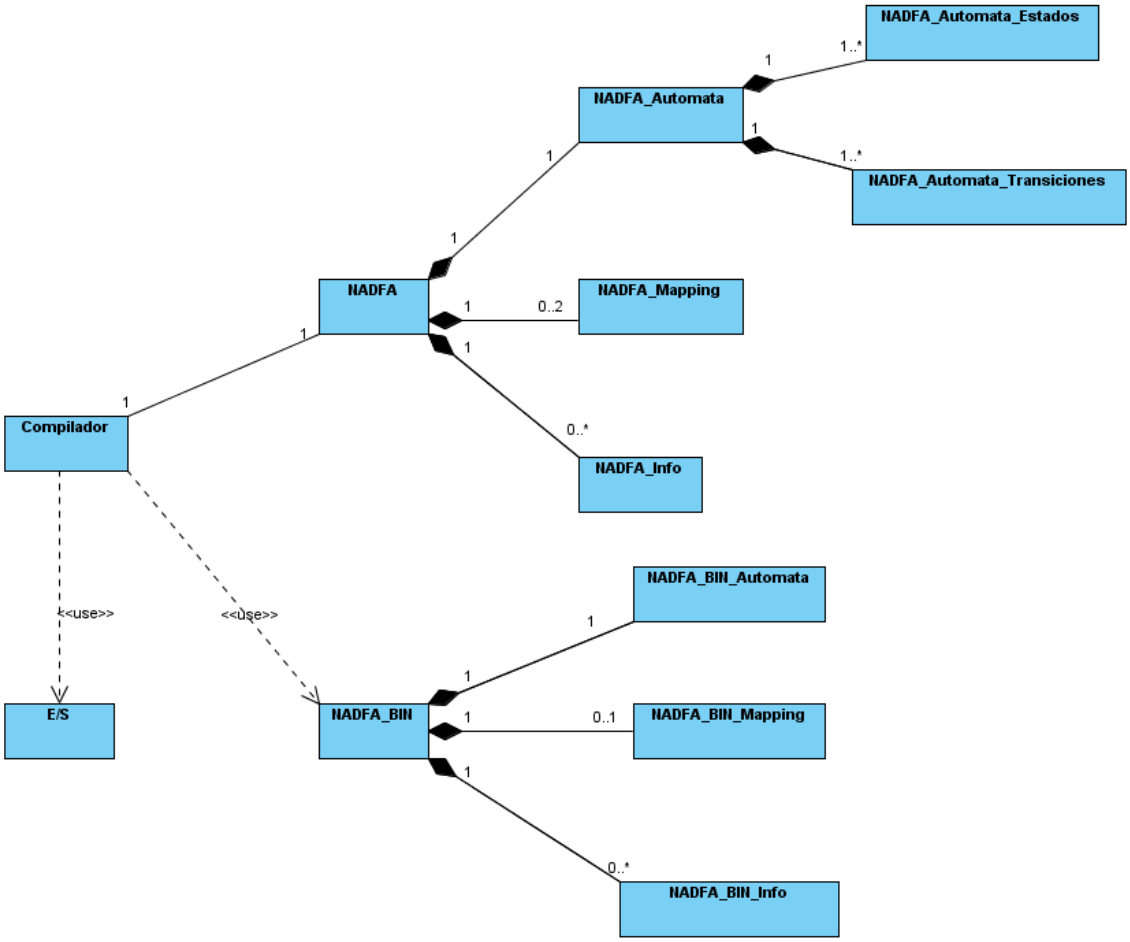

Figura 38. Diagrama de Clases del sistema (Análisis).

DISEÑO

En la fase de diseño se estudia cómo deben construirse las funcionalidades del sistema para satisfacer los requisitos iniciales.

1. DIAGRAMA DE CLASES

Las dos partes del sistema diferenciadas en el análisis como “Compilación Diccionario” y “Acceso Diccionario Compilado” aparecen reflejadas también en el diagrama de clases, por un

lado la parte que controla la clase “Compilador” (Compilación Diccionario) y por otro la parte que controla la clase “NADFA_BIN” (Acceso Diccionario Compilado).

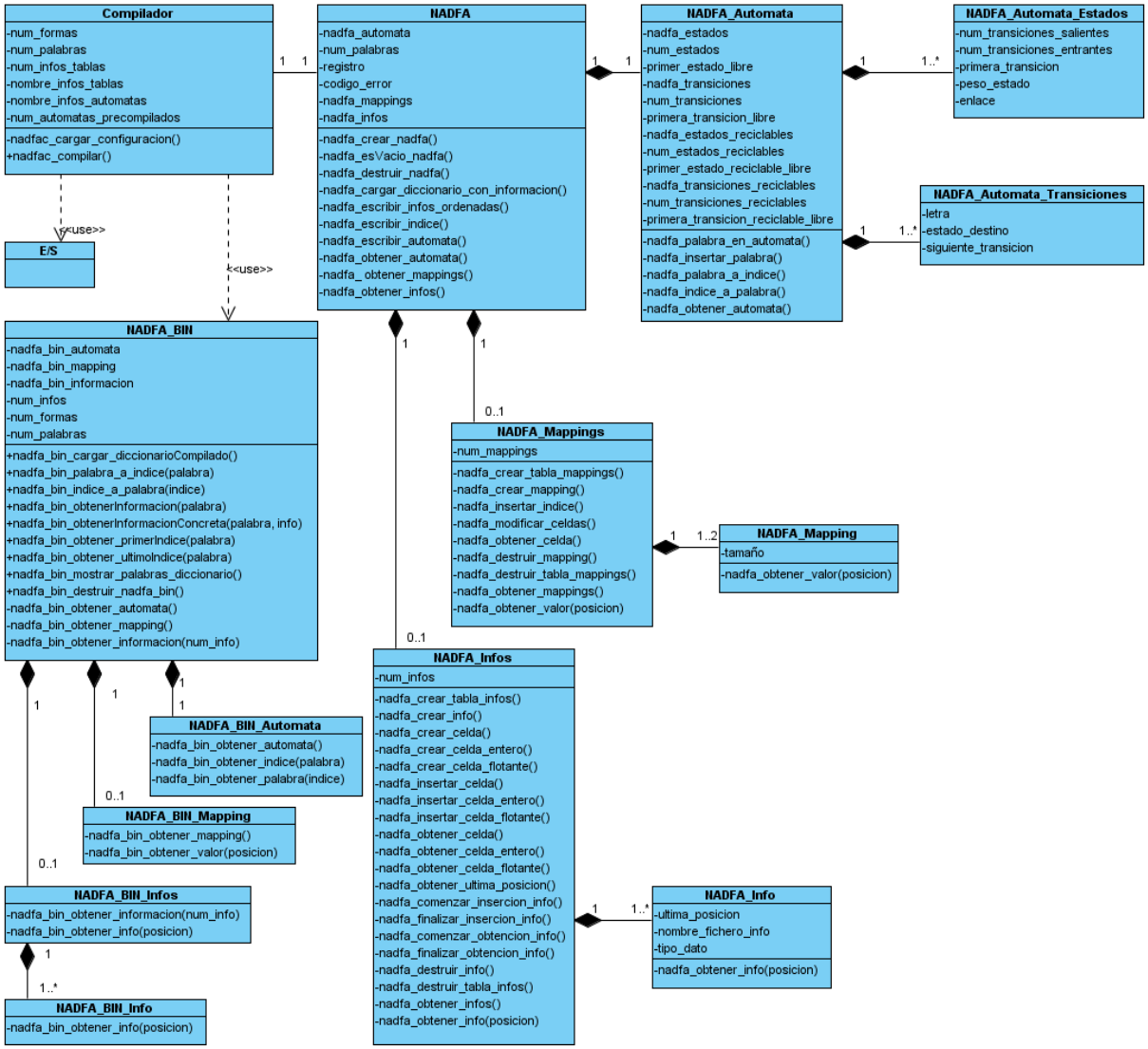

Figura 39. Diagrama de Clases del sistema (Diseño).

2. DICCIONARIO DE CLASES

2.1 Compilador

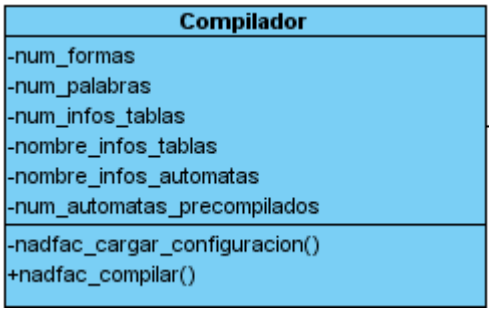

Figura 40. Diagrama de clases parcial de la clase Compilador.

El compilador es la clase encargada de llevar a cabo todo el proceso de compilación de un diccionario. Interactúa con la clase NADFA para poder crear el autómata (que permite almacenar las palabras del diccionario), las tablas que almacenan las columnas de información y los *mappings* que permiten acceder a la información.

Mediante la clase E/S el compilador accede al fichero XML de configuración para obtener los parámetros necesarios para llevar a cabo las operaciones y también los nombres y la localización del resto de ficheros a los que debe acceder. También interactúa con la clase NADFA_BIN para poder acceder a los diccionarios previamente compilados, en caso de ser necesario para generar el principal.

Los atributos y métodos de esta clase se muestran a continuación:

ATRIBUTOS
num_formas: número de formas[4] que contiene el diccionario.
num_palabras: número total de palabras que contiene el diccionario.
num_infos_tablas: número de columnas de información del diccionario.
nombre_infos_tablas: *array* que almacena los nombres de las columnas de información del diccionario.
nombre_infos_automatas: *array* que almacena los nombres de las columnas de información cuyas entradas provienen de un diccionario previamente compilado. Cada nombre lo almacena en el índice correspondiente al identificador de dicho diccionario.
num_automatas_precompilados: número de diccionarios previamente compilados.
MÉTODOS
nadfac_cargar_configuracion(): toma los datos del fichero de configuración y los carga en una estructura.
nadfac_compilar(): lleva a cabo todas las operaciones necesarias para compilar el diccionario.

[4] La forma léxica de una palabra es la propia palabra tal y como se escribe, la forma gráfica. Diferenciar de lema que es la "forma canónica" de la palabra. Un lema representa un conjunto de palabras con la misma raíz, misma categoría léxica (tipo de palabra) y mismo significado. Normalmente los lemas coinciden con las entradas de un diccionario convencional (en papel): el lema de un verbo es su infinitivo y el de un sustantivo el masculino singular.

El compilador comprime un diccionario de palabras junto con la información asociada a cada una de ellas. Se ha seguido el algoritmo de Daciuk [2] para la inserción de las palabras del diccionario, y la implementación de Graña [3] para comprimir la información asociada, generalizando la visión de este último a un diccionario de palabras que no tienen por qué estar ordenadas alfabéticamente, con cualquier número de columnas de información y con el empleo de autómatas previamente compilados para aquellas columnas de información en las que sea necesario.

Para poder almacenar las palabras del diccionario se parte de un fichero XML de configuración que permite saber la ruta en la que se encuentran todos los ficheros necesarios para la compilación y el manejo de la librería.

El compilador recoge toda la información del fichero de configuración XML y la carga en memoria en una estructura que permitirá el fácil y rápido acceso a dicha información durante el proceso de compilación. Seguidamente el sistema accede al fichero que contiene la lista de palabras del diccionario y hace un recuento de la cantidad de palabras que hay y el número de formas.

A continuación, el sistema pasa al proceso de compilación propiamente dicho, cuyo fin es comprimir el diccionario en un fichero binario. Para ello se emplea la clase NADFA que primero lee línea a línea el diccionario y carga en memoria:

- el autómata (en el que se insertan la palabras del diccionario),
- las tablas *mapping* con los índices de acceso a la información *y*
- las tablas de información de cada palabra (recopilando las entradas de cada columna ó el índice correspondiente a su autómata previamente compilado, en caso de tener uno asociado).

Una vez se tiene el diccionario cargado en memoria se comprime el autómata y se escribe en el fichero binario de salida, se unifican las tablas *mapping* y se escribe el índice resultante y se finaliza con la escritura de las tablas de información de forma ordenada según el orden alfabético de las palabras del autómata, obteniendo así la versión compilada del diccionario de partida.

2.2 NADFA

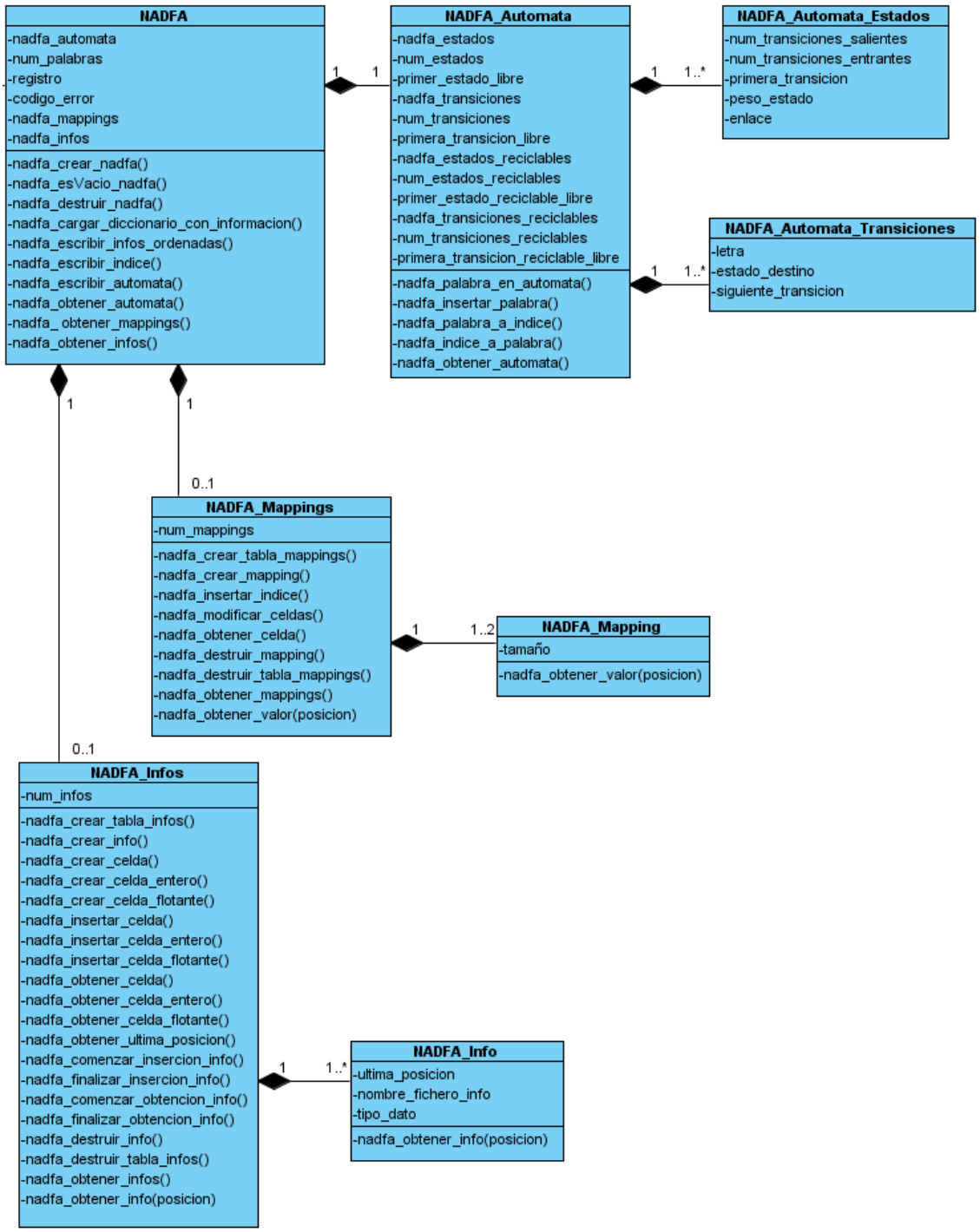

Figura 41. Diagrama de clases parcial de la clase NADFA.

La clase NADFA gestiona conjuntamente los objetos Autómata, Mapping e Info. Representa el diccionario en su totalidad y permite acceder al autómata que almacena las palabras, a las columnas de información de cada palabra del diccionario y a los índices a las entradas de dichas columnas cargados en memoria.

La clase NADFA_Automata representa el autómata finito determinista minimizado que almacena el conjunto de palabras del diccionario. Los estados y transiciones del autómata se representan mediante las clases NADFA_Automata_Transiciones y NADFA_Automata_Estados.

La clase NADFA_Mapping representa la tabla de conversión que, a partir de un índice localiza las entradas de información de cada palabra.

La clase NADFA_Info representa una columna de información de las palabras.

ATRIBUTOS (NADFA)
nadfa_automata: autómata para las palabras del diccionario. Se representa mediante objetos de la clase NADFA_Automata.
num_palabras: número de palabras del diccionario.
registro: contiene en todo momento los estados que pertenecen al autómata mínimo (necesario para implementar el algoritmo de Daciuk).
código_error: número identificativo del error surgido, en caso de existir alguno, en las llamadas a la librería.
nadfa_mappings: tabla de los índices a la información de las palabras. Se representa mediante objetos de la clase NADFA_Mappings.
nadfa_infos: tabla de las columnas de información de las palabras. Se representa mediante objetos de la clase NADFA_Infos.
MÉTODOS (NADFA)
nadfa_crear_nadfa(): crea un objeto de la clase NADFA.
nadfa_esVacio_nadfa(): permite saber si un objeto NADFA está vacío.
nadfa_destruir_nadfa(): permite eliminar un objeto de la clase NADFA.
nadfac_cargar_diccionario_con_informacion(): crea el autómata e inserta las palabras en él, crea las tablas de información e inserta los valores con un enlace a la posición de la siguiente entrada para cada palabra (en caso de haber más de una), crea las tablas *mapping* e introduce los índices de las entradas de información.
nadfa_escribir_infos_ordenadas(): escribe en el fichero binario de salida cada columna de información según el orden alfabético correspondiente a las palabras del diccionario, de modo que la información para palabras con la misma forma está en posiciones contiguas.
nadfa_escribir_indice(): unifica en un solo índice las tablas *mapping* almacenando en cada posición el índice de la información correspondiente a la primera forma para cada palabra (la información ya ordenada) y lo escribe en el fichero binario de salida.
nadfa_escribir_automata(): comprime el autómata a nivel de bit y escribe en el fichero binario de salida la versión comprimida con las palabras del diccionario.
nadfa_obtener_automata(): permite acceder a los elementos del autómata.
nadfa_obtener_mappings(): permite acceder a los elementos de las tablas *mapping.*
nadfa_obtener_infos(): permite acceder a los elementos de las tablas de información.

2.2.1 NADFA_Automata

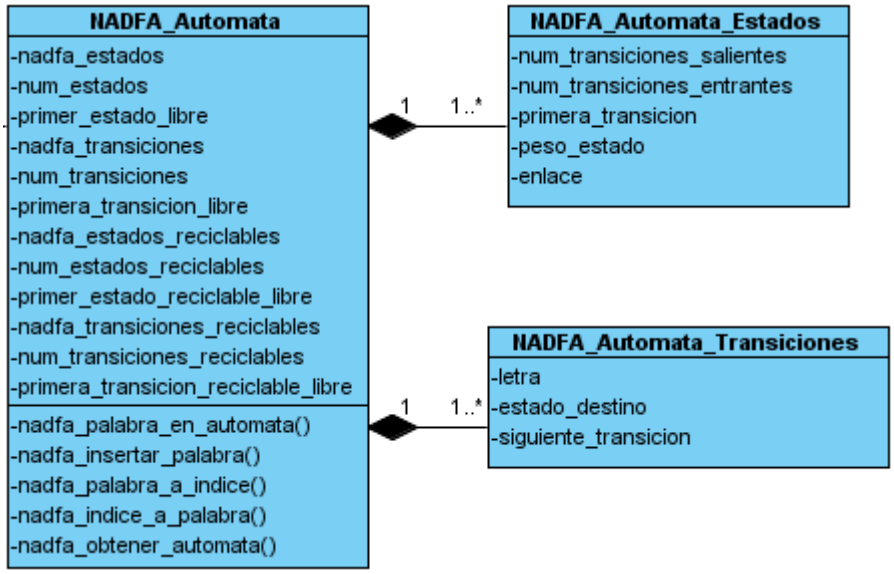

Figura 42. Diagrama de clases parcial de la clase NADFA_Automata

ATRIBUTOS (NADFA_Automata)
nadfa_estados: conjunto de estados del autómata. Se representa mediante objetos de la clase NADFA_Automata_Estados.
num_estados: número de estados del autómata.
primer_estado_libre: primera posición libre en el *array* de estados del autómata.
nadfa_transiciones: conjunto del conjunto de transiciones del autómata. Se representa mediante objetos de la clase NADFA_Automata_Transiciones.
num_transiciones: número de transiciones del autómata.
primera_transicion_libre: primera posición libre en el *array* de transiciones del autómata.
nadfa_estados_reciclables: conjunto de estados posiblemente no útiles para el autómata minimizado.
num_estados_reciclables: número de estados reciclables.
primer_estado_reciclable_libre: primera posición libre en el *array* de estados reciclables.
nadfa_transiciones_reciclables: conjunto de transiciones posiblemente no útiles para el autómata minimizado.
num_transiciones_reciclables: número de transiciones reciclables.
primera_transicion_reciclable_libre: primera posición libre en el *array* de transiciones reciclables del autómata.
MÉTODOS (NADFA_Automata)
nadfa_palabra_en_automata(): comprueba si una palabra concreta es reconocida por el autómata.
nadfa_insertar_palabra(): permite la inserción de una palabra en el autómata.
nadfa_palabra_a_indice(): permite la conversión de una palabra a su índice en el autómata.
nadfa_indice_a_palabra(): permite la conversión de un índice del autómata a la palabra correspondiente.
nadfa_obtener_automata(): permite acceder a los elementos del autómata.

ATRIBUTOS (NADFA_Automata_Estados)
num_transiciones_salientes: número de transiciones que salen del estado.
num_transiciones_entrantes: número de transiciones que llegan al estado.
primera_transicion: primera transición a atravesar por el estado.
peso_estado: número de subcadenas reconocibles por el estado.
enlace: posición del estado en el *array* de estados.

ATRIBUTOS (NADFA_Automata_Transiciones)
letra: carácter de la transición.
estado_destino: estado al que llega la transición.
siguiente_transicion: siguiente transición en el conjunto de transiciones del autómata.

Construcción

Tradicionalmente, para construir un diccionario mínimo (un autómata mínimo que reconoce un conjunto finito de palabras) se construía primero un diccionario no necesariamente mínimo (a través de, por ejemplo, un árbol de letras), y después se minimizaba. El problema de esta técnica es que el diccionario inicial sobre el que se debe aplicar la operación de minimización puede ocupar demasiado espacio en memoria, lo que puede hacer que el algoritmo no se pueda llevar a la práctica en casos donde se necesitan manejar diccionarios considerablemente grandes.

Por el contrario, el algoritmo propuesto por Daciuk [2] minimiza la utilización de memoria mediante la construcción del autómata mínimo incrementalmente (palabra a palabra y manteniendo el autómata siempre mínimo).

En el artículo de Daciuk [2] se presentan dos algoritmos, uno donde las palabras se insertan en el autómata en orden lexicográfico, y otro donde las palabras llegan en un cualquier orden. Debido a que queremos que nuestra librería sea lo más genérica posible, hemos implementado el algoritmo de inserción desordenada.

La mayoría de los algoritmos de minimización de autómatas se basan en realizar una clasificación de los estados de un árbol o un autómata no mínimo que representan el diccionario. Éstos se particionan en clases de equivalencia, que configurarán los estados del autómata mínimo (es decir, por cada clase de equivalencia habrá un estado en el autómata mínimo). Además, podemos afirmar que en el autómata mínimo no existen dos estados que tienen el mismo lenguaje por la derecha, y esta afirmación es condición necesaria y suficiente para la minimalidad, por lo que se puede utilizar este criterio para calcular las clases de equivalencia (los estados del autómata mínimo).

El algoritmo de minimización utiliza un registro, que contiene en todo momento los estados que pertenecen al autómata mínimo (las clases de equivalencia). En este se lleva a cabo la función “reemplazar o registrar” que, dado un estado, comprueba si hay otro equivalente en el registro. En caso afirmativo, elimina el primero, y hace que las transiciones entrantes al mismo apunten al estado equivalente (operación de reemplazar). Por el contrario, en caso de que no

haya un estado equivalente en el registro, introduce el estado en el registro (operación de registrar).

Para obtener una mayor claridad en las explicaciones, se supondrá que siempre habrá un carácter más en las palabras (un carácter que indica el fin de palabra), y se representará con el símbolo #. De esta forma se consigue que en el autómata solamente haya un único estado final, simplificando así algunas operaciones del algoritmo.

Suponer la existencia de dicho carácter como fin de palabra, implica que cada vez que se añada un sufijo, el último estado que se crea siempre va a ser equivalente al estado final, con lo que podemos evitar realizar la operación de reemplazar o registrar con estos estados finales del sufijo, simplemente uniendo el penúltimo estado creado con la última transición al estado final, consiguiendo así una mayor eficiencia.

La consideración anterior provoca que el estado final debe estar registrado antes de insertar una palabra (incluida la primera), por lo que lo primero que se debe hacer cuando se crea el autómata es registrar el estado final. De igual modo también se registra el estado inicial, ya que también es único, es decir el autómata vacío no es aquel que no tiene ningún estado, sino que es un autómata con los estados inicial y final creados y registrados (aunque inconexos).

Asimismo, en los diagramas el estado final siempre será el estado número 1, dejando el número 2 para el estado inicial, y números consecutivos a partir del 3 para los estados que se van creando.

A continuación se muestran los pasos del algoritmo de inserción implementado:

- Para cada palabra a insertar:
  - Calcular el prefijo común: que es la subcadena más larga de la palabra a insertar que puede avanzar a través de las transiciones del autómata.
  - Calcular ultimo_estado: que es el estado al que se llega con el prefijo común.
  - Calcular el primer estado confluencia.
  - Si no hay primer estado confluencia:
    - Sacar ultimo_estado del registro.
    - Añadir el sufijo (la parte de la palabra no incluída en el prefijo común) a ultimo_estado, reemplazando o registrando los estados del sufijo.
  - Si hay primer estado confluencia:
    - Clonar ultimo_estado.
    - Añadir el sufijo al clon, reemplazando o registrando los estados del sufijo.
    - Calcular nuevamente el primer estado confluencia.
    - Sacar del registro el estado anterior al primer confluencia.
    - Para todos los estados del prefijo común desde el anterior a ultimo_estado hasta el primer confluencia:
      - Crear un clon.
      - Añadir una transición desde este clon hasta el clon creado anteriormente (la primera vez será el clon de ultimo_estado).
      - Reemplazar o registrar el estado que se ha clonado anteriormente.

- Unir el estado anterior al confluencia con el último clon creado y reemplazar o registrar este último clon.

- Para los estados del prefijo común no procesados y hasta que no haya un reemplazo (en el caso de que haya habido un primer confluencia, se empieza con el estado anterior al primer confluencia, y en el caso de que no lo haya habido, se comienza con ultimo_estado):
  - Reemplazar o registrar el estado a procesar.
  - Si la operación anterior resulta en un reemplazo, sacar del registro el estado anterior (el que será procesado a continuación).

El algoritmo permite el control de la aparición de ciclos. Cuando se quiere insertar una palabra cuyo sufijo (la parte de la palabra que no está en el prefijo común) tiene una parte final que coincide con el sufijo de alguna palabra que ya se ha introducido en el autómata, pueden aparecer ciclos, provocando un malfuncionamiento en la construcción del algoritmo.

Teniendo en cuenta el modo en que se introducen los estados en el registro (siempre primero los hijos de un estado antes que el propio estado), es posible solucionar el problema simplemente sacando ultimo_estado del registro antes de insertar el sufijo en el mismo (y por tanto antes de reemplazar o registrar los estados del sufijo). Al quitar ultimo_estado del registro antes de insertar el sufijo se consigue que no aparezcan estados equivalentes en todo el camino hasta éste, y por tanto no se generarán ciclos.

A continuación puede observarse este proceso, partiendo del autómata de la figura 43, si se realiza la inserción de la palabra “abcdecd”, sin control de ciclos resultaría el autómata de la figura 44, mientras que con control de ciclos se obtiene el de la figura 45.

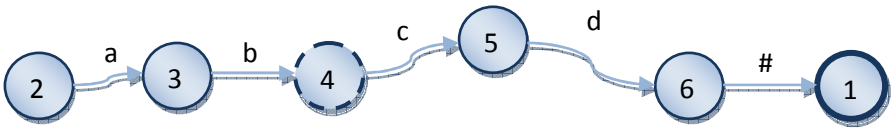

Figura 43. Autómata inicial que reconoce la cadena "abcd"

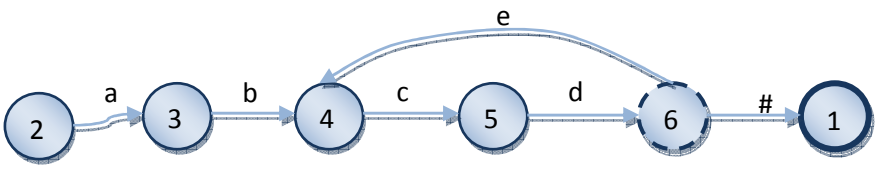

Figura 44. Inserción de la palabra "abcdecd" sin control de ciclos

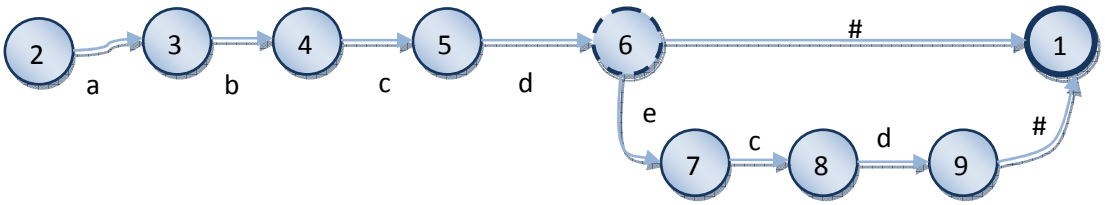

Figura 45. Inserción de la palabra "abcdecd" con control de ciclos.

Para que el autómata sea mínimo no llega con reemplazar o registrar ultimo_estado, si no que es necesario ir sacando del registro y reemplazando o registrando los estados que pertenecen al camino del prefijo común hasta llegar al estado inicial o hasta que haya un estado que no es reemplazado (en este caso no es necesario seguir chequeando los estados anteriores, porque si un estado no tiene equivalente en el registro, un estado que apunte a éste tampoco, y así sucesivamente hasta llegar al estado inicial).

El algoritmo también controla la aparición de un estado confluencia, es decir, un estado que tiene más de una transición entrante, y que por lo tanto podría provocar que se insertaran palabras no deseadas. Para evitar este tipo de problemas se emplea la operación “clonación”: proceso de crear un nuevo estado con exactamente las mismas transiciones salientes que el estado original (el estado a clonar). Esta operación permite aislar la parte del autómata sobre la que se va a hacer la inserción de la nueva palabra y se evita la generación de palabras indeseadas.

A continuación puede observarse este proceso, partiendo del autómata de la figura 46, si se realiza la inserción de la palabra “abcsn”, sin control de estados confluencia resultaría el autómata de la figura 47, mientras que con control de estados confluencia se obtiene el de la figura 48, que reconoce las palabras deseadas.

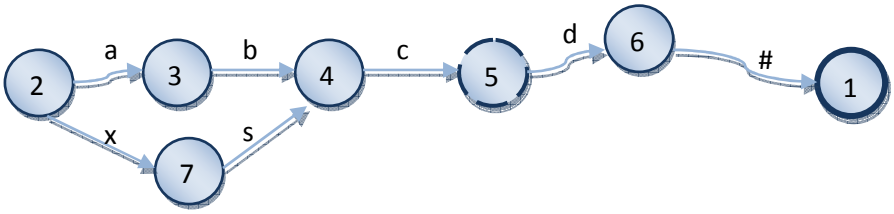

Figura 46. Autómata inicial que reconoce la cadena "abcd" y “xscd”.

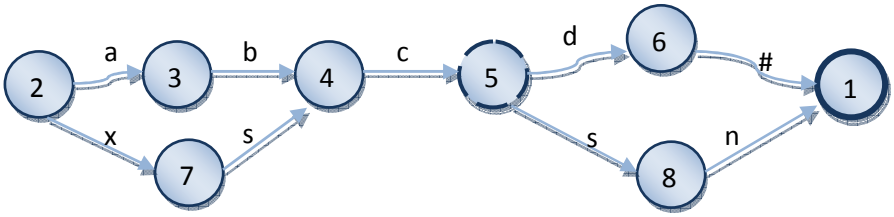

Figura 47. Inserción de la palabra "abcsn" sin control de estados confluencia. No solamente se reconoce “abcsn”, sino también “xscsn”.

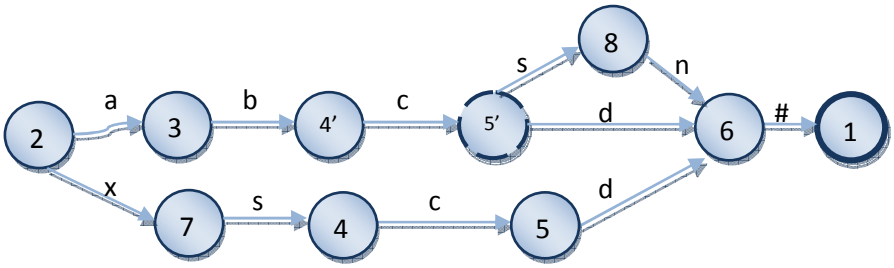

Figura 48. Inserción de la palabra "abcsn" con control de estados confluencia. Reconoce "abcsn", "abcd" y "xscd".

De este modo el algoritmo permite añadir palabras para la creación de un autómata finito acíclico mínimo, para que el autómata sea numerado se asigna un peso a cada uno de los estados, y un valor numérico único a cada una de las palabras, permitiendo hacer un "*hashing* perfecto" palabra-identificador. Este peso indica el número de subcadenas que se reconocen desde un estado al estado final. Por lo tanto, hay que añadir al algoritmo anterior:

- En la creación e inicialización del autómata, se asigna a los estados inicial y final peso 0.
- Cuando se crea un estado se le asigna peso 1.
- Cuando se clona un estado, se le asigna el peso que tiene el estado original a clonar.
- En el proceso de clonación de estados desde ultimo_estado hasta el primer confluencia, justo antes de reemplazar o registrar un estado clon hay que sumar uno a su peso.
- Cuando en la fase final se procesan los estados del prefijo común que no se han procesado, hay que ir añadiendo uno al peso justo antes de reemplazarlo o registrarlo nuevamente. Dado que hay que actualizar el peso de todos los estados hasta el estado inicial, hay que seguir el proceso hasta el estado inicial, y por lo tanto no se puede parar el procesamiento en el caso de que no se produzca un reemplazo tal y como se ha descrito anteriormente.

Más abajo se muestra el pseudocódigo detallado del algoritmo para la inserción de palabras en el autómata. Debe tenerse en cuenta que se asume, como se ha mencionado anteriormente, que existe un carácter de fin de cadena al final de la palabra, y éste será un carácter no válido como elemento de la cadena, es decir ese carácter no puede aparecer en ninguna palabra en otra posición que no sea al final de la cadena.

Por otra parte, dado que el estado inicial y el estado final serán únicos (nunca se generará un equivalente a éstos), asumimos que no están en el registro, es decir, el autómata mínimo estará formado por los estados del registro más los estados inicial y final. Con esto evitamos complicar el pseudocódigo.

Como se ha podido comprobar, una de las operaciones fundamentales del autómata es la de reemplazar o registrar un estado. Dado que cuando se produce un reemplazo hay que cambiar las transiciones entrantes que llegan al estado que se está chequeando para que apunten al

estado equivalente, y que siempre se intentan reemplazar o registrar estados que solamente tienen una transición entrante, la función **reemplaza_o_registra** recibe el estado padre del estado que se chequea y el carácter con el que se llega al hijo, de manera que se tienen todos los elementos necesarios para realizar la operación, tanto si resulta en un reemplazo como en un registro.

```
PROCEDIMIENTO reemplaza_o_registra(estado_padre, caracter)
INICIO
    hijo ← transita(estado_padre, caracter)
    equivalente ← estado_equivalente(hijo)
    SI equivalente <> vacio
        elimina_estado(hijo)
        cambio_transicion(estado_padre,carácter,equivalente)
    SINO
        registra(hijo)
    FIN_SI
FIN_PROCEDIMIENTO
```

La función **transita** devuelve el estado destino que se obtiene al transitar desde el estado que se le pasa con el carácter, la función **estado_equivalente** devuelve el estado equivalente del registro a uno dado (o vacío si no hay un estado equivalente en el registro), **elimina_estado** elimina el estado, **cambia_transicion** cambia la transición destino correspondiente al carácter, del estado padre al estado equivalente, y **registra** mete el estado que recibe en el registro (es la operación de registrar propiamente dicha).
Por lo tanto, el algoritmo para añadir una palabra se divide en 3 fases. En la primera de ellas se trata el cálculo de ultimo_estado, el prefijo común y la inserción del sufijo, en la segunda se trata de la clonación desde ultimo_estado hasta el primer confluencia, y en la tercera se afronta la parte de tratar los estados del prefijo común aún no procesados, quitándolos del registro y reemplazándolos o registrándolos nuevamente.

Los nombre de las funciones son auto-explicativos, y reseñaremos la función **transita_varios**, que simplemente intenta avanzar en el autómata con todos los caracteres de la subcadena que recibe (en lugar de con un solo carácter, como en el caso de la función transita).

```
FUNCION añade_palabra(palabra) {
INICIO
        estado_inicial ← obten_estado_inicial()
        estado_final ← obten_estado_final()
        prefijo_comun ← calcula_prefijo_comun(palabra)
        ultimo_estado ← transita_varios(estado_inicial, prefijo_comun)
        sufijo ← palabra[longitud(prefijo_comun)..longitud(palabra)-1]
        SI sufijo = vacio Y es_estado_final(ultimo_estado)
                DEVOLVER Falso
        FIN_SI
        primer_confluencia ← calcula_primer_confluencia(estado_inicial, prefijo_comun)
        SI primer_confluencia = vacio
                SI ultimo_estado <> estado_inicial
                        elimina_del_registro(ultimo_estado)
                FIN_SI
        SINO
                ultimo_estado ← clona (ultimo_estado)
        FIN_SI
        añade_sufijo (ultimo_estado, sufijo)
FIN_FUNCION
```

Para realizar una implementación óptima del pseudocódigo, debe tenerse en cuenta lo siguiente:

- Existen algunas funciones que pueden realizarse conjuntamente. Por ejemplo, se puede calcular ultimo_estado y el prefijo común dentro de la misma función, o el primer (estado) confluencia y el anterior al primer confluencia, evitando así navegar por el autómata más veces de las necesarias.
- Se ha asumido que hay un carácter de fin de cadena para las palabras. Esto puede suponer dos problemas:
  - La implementación no quedaría muy elegante, ya que habría que añadir un carácter a la palabra que se desea insertar.
  - Se podría necesitar que ese carácter sea válido formando parte de las palabras, y el algoritmo no sería válido.

Para solucionar lo primero, se puede gestionar ese carácter internamente, es decir en lugar de añadir el carácter al final de la palabra, se puede gestionar internamente en la lógica de la aplicación. De esta manera se puede utilizar esto para solucionar también el segundo problema, eligiendo como carácter de fin de cadena el carácter 0, que es un carácter no imprimible y que además ya indica el fin de cadena en muchos lenguajes, consiguiendo así que todos los caracteres imprimibles sean válidos para formar parte de las palabras.

Uno de los puntos críticos en la implementación del algoritmo, y que determina la complejidad del mismo, es la implementación del registro. Existen dos funciones que acceden al registro,

por un lado **reemplaza_o_registra**, y por el otro **elimina_del_registro**. Es conveniente que se pueda buscar un estado al menor coste posible (elimina_del_registro), y también es necesario que se pueda averiguar el estado equivalente a uno dado y que sea posible registrarlo (reemplazar_o_registrar). Para buscar o insertar rápidamente un estado en el registro, podemos implementar el registro como una tabla hash, donde la clave es el identificador del estado, consiguiendo así una complejidad ideal constante.

Sin embargo, para poder localizar rápidamente un estado equivalente a uno dado es necesario que la clave incorpore otras características del nodo a almacenar (las transiciones que tiene y a qué estados), con lo que la función hash debería considerar propiedades adicionales de los estados para generar la clave correspondiente, Hay que tener en cuenta que una consulta a la tabla hash nos devolverá un conjunto de estados candidatos a equivalente, por lo que habría que comparar con los valores de los estados reales, y se conseguiría una complejidad ideal constante en las dos funciones críticas del algoritmo.

A la hora de insertar una nueva transición en un estado, ha y dos alternativas: añadirla como última transición del estado o meterla en el lugar correspondiente según el orden lexicográfico de los caracteres de transición. La ventana de escoger la segunda alternativa es que las palabras se encuentran lexicográficamente ordenadas, de manera que se podrían localizar palabras menores o mayores lexicográficamente a una dada, es un poco menos eficiente la inserción, pero resulta un poco más eficiente el acceso y la búsqueda.

Una vez se ha construido el autómata numerado, las funciones **palabra_a_índice** e **índice_a_palabra** realizan la conversión biunívoca de una palabra a su identificador correspondiente y viceversa.

La función **se_puede_transitar** indica si hay una transición desde el estado que se le pasa con el carácter. La función **caracter_primera_transicion** devuelve el carácter correspondiente a la primera transición del estado, y la función **caracter_anterior_transisicion** devuelve el carácter de la transición anterior a la correspondiente al carácter con el que se llama.

La idea del algoritmo correspondiente a **palabra_a_índice** es ir avanzando en el autómata con los caracteres de la palabra, e ir sumando los pesos de los estados correspondientes a las transiciones destino que se van descartando anteriores a la que se escoge, mientras que **índice_a_palabra** se basa en ir restando del valor del índice los pesos de los estados destino de las transiciones de un estado hasta que el valor del peso del estado destino que se chequea sea mayor que el valor actual, en cuyo caso se avanza por dicha transición y la letra de la palabra correspondiente es la de la misma, continuando con este proceso hasta que se llega a un estado final y el valor del índice es cero.

```
FUNCION palabra_a_indice(palabra)
INICIO
        indice ← 1
        estado_inicial ← obten_estado_inicial()
        estado_actual ← estado_inicial
        DESDE contador ← 0 HASTA longitud(palabra)
                SI se_puede_transitar (estado_actual, palabra[contador])
                        DESDE caracter ← caracter_primera_transicion(estado_actual) HASTA
                        caracter_anterior(estado_actual, palabra[contador])
                                indice = indice + peso(transita(estado_actual, caracter))
                        FIN_DESDE
                        estado_actual ← transita (estado_actual, palabra[contador])
                SINO
                        DEVOLVER 0
                FIN_SI
        FIN_DESDE
        SI es_estado_final(estado_actual)
                DEVOLVER indice
        SINO
                DEVOLVER 0
        FIN_SI
FIN_FUNCION
```

Debe tenerse en cuenta que, dado que las palabras pueden llegar en cualquier orden, la construcción del autómata se lleva a cabo según el orden de llegada (estén o no ordenadas las palabras), por lo que las correspondencias palabra-identificador no se mantienen, es decir, el identificador de las palabras que ya había en el autómata y posteriores a la que inserta, cambiarán (identificadores mayores al último calculado). Al abrir hueco para una nueva palabra se reorganizan los índices, como puede observarse en la siguiente figura. Con la inserción de la palabra "casa" la palabra que antes tenía el índice 2 ahora pasa a tener el índice 3, por lo que las palabras siguientes también pasan a incrementar en 1 su índice:

Figura 49. Reorganización de índices en inserción de palabra.

```
FUNCION indice_a_palabra(indice)
INICIO
        estado_inicial ← obten_estado_inicial()
        estado_actual ← estado_inicial
        numero_auxiliar ← indice
        contador ← 0
        palabra ← palabra_vacia()
        MIENTRAS numero_auxiliar > 0
                DESDE carácter ← caracter_primera_transicion(estado_actual) HASTA
                caracter_ultima_transicion(estado_actual)
                        SI numero_auxiliar > peso(transita(estado_actual, caracter))
                                SI es_estado_final(transita(estado_actual, caracter))
                                        numero_auxiliar ← numero_auxiliar - 1
                                        SI numero_auxiliar = 0
                                                INTERRUMPIR
                                        SINO
                                                numero_auxiliar ← numero_auxiliar –
                                                peso(transita(estado_actual, caracter))
                                        FIN_SI
                                SINO
                                        palabra[contador] ← caracter
                                        contador ← contador + 1
                                        estado_actual ← transita (estado_actual, caracter)
                                        INTERRUMPIR
                                FIN_SI
                        FIN_SI
                        SI numero_auxiliar = 0 Y
                        transita(estado_actual, caracter_final) = estado_final()
                                palabra[contador] ← caracter_final
                                DEVOLVER palabra
                        SINO
                                DEVOLVER vacio
                        FIN_SI
                FIN_DESDE
        FIN_MIENTRAS
FIN_FUNCION
```

Escritura

Por lo tanto, una vez construido el autómata, el compilador necesitará almacenarlo de manera compilada en disco del siguiente modo:

Casilla estado (4 bytes): cada estado almacena su número de transiciones salientes en 1 byte, el de mayor peso, y en los 3 bytes restantes almacena el peso de dicho estado.

Casilla transición (4 bytes): cada transición almacena la letra que transita en su byte de mayor peso, y en los 3 bytes restantes almacena la posición (dentro de la estructura que almacena el autómata en memoria) en la que está el estado al que llega dicha transición.

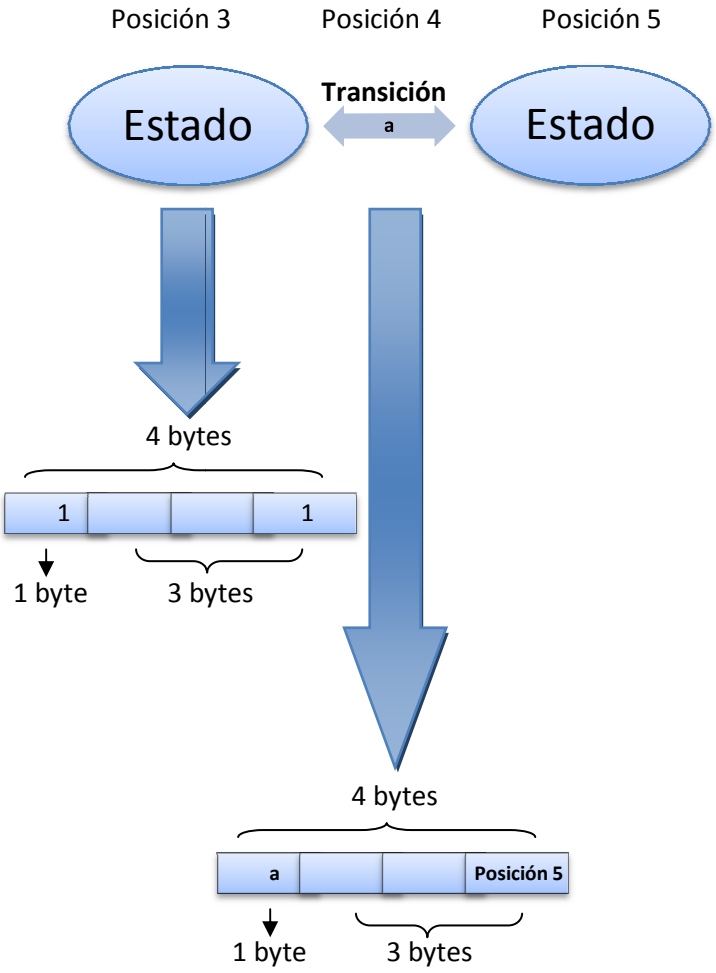

Figura 50. Distribución de bytes en el autómata

En la figura 51 se muestra el formato de la estructura comprimida del autómata.

Finalizado el proceso de compilación se pasa a escribir todo en el fichero binario de salida:

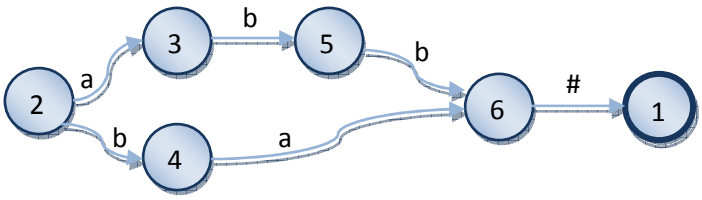

Figura 51. Autómata inicial que reconoce la cadena "abb" y "ba".

Primero se escribe el número total de estados y transiciones que tiene el autómata para saber el número de casillas que ocupará cuando se utilice. Esta cantidad se obtiene sumando 1 por cada estado y por cada transición de dicho estado.

Seguidamente se escribe el estado final (Estado1)con 0 transiciones y peso cero.

A partir de ahí se escribe ordenadamente los estados siguiendo el índice del *array* de estados en el que están almacenados, y por cada estado:

- Se construye una celda estado (un entero de 4 bytes) y se almacena en esa celda el número de transiciones salientes para ese estado.
  celda_estado = número de transiciones salientes del estado
- Mediante el operador de desplazamiento de bits[5] (<<) se desplazan hacia la izquierda los 8 bits de menor peso que contienen el número de transiciones salientes y se almacena en los 24 bits restantes (a la derecha) el peso[6] del estado mediante el operador de suma (+), de modo que ya está completada la celda estado.
  celda_estado <<= 24
  celda_estado += peso del estado
- Se escribe en fichero la celda estado creada y por cada transición del estado:
  - Se crea una celda transición (un entero de 4 bytes) y se almacena en esa celda el carácter de esa transición.
    celda_transición = letra que transita
  - Mediante el operador de desplazamiento de bits (<<) se desplazan hacia la izquierda los 8 bits de menor peso que contienen la letra de la transición y se almacena en los 24 bits restantes (a la derecha) la posición del estado al que llega la transición mediante el operador de suma (+), de modo que está completada la celda transición.
    celda_transición <<= 24
    celda_transición += posición del estado al que llega
  - Se escribe en fichero la celda transición creada.

[5] La implementación está diseñada para una arquitectura *BIG ENDIAN* (el de la izquierda es el byte más significativo y el de la derecha el menos significativo).
[6] Nº de subcadenas reconocibles por el autómata a partir del estado actual.

Escritura en fichero:

0	1	2	3	4	5	6	7	8	9	10	11	12
12	0	2	a	b	1	b	1	a	1	b	1	#
	0	2	5	7	1	9	1	11	1	11	1	0
	Estado1		Estado2		Estado3		Estado4		Estado5		Estado6	

nº de celdas del autómata
nº transiciones salientes del estado
peso del estado
letra de la transición
puntero al estado destino de la transición
posición en el fichero binario

Figura 52. Estructura comprimida del autómata.

Puede verse el proceso de escritura mediante el siguiente pseudocódigo, para el que se han empleado llamadas a funciones y métodos con nombres auto-explicativos. La función **almacena** lleva a cabo el proceso de comprimir tanto una celda de un estado (línea 17) como de una transición (línea 23) que en la solución implementada se resuelve con los operadores de desplazamiento (<<) y or lógico (+) que se acaban de explicar.

```
PROCEDIMIENTO escribir_automata()
INICIO
        indice_celda ← 2
        primer_estado_libre ← obten_primer_estado_libre()
        DESDE indice_estado ← 2 HASTA primer_estado_libre
                almacena_enlace (indice_estado, indice_celda)
                num_transiciones ← num_transiciones_salientes(indice_estado)
                SI num_transiciones <> 0
                        indice_celda ← indice_celda + num_transiciones + 1
                FIN_SI
                escribe_en_fichero(indice_celda)
                celda ← 0
                escribe_en_fichero(celda)
                DESDE indice_estado ← 2 HASTA primer_estado_libre
                        almacena(num_transiciones_salientes(indice_estado),
                        peso(indice_estado),celda)
                        escribe_en_fichero(celda)
                        indice_transicion = primera_transicion(indice_estado)
                        MIENTRAS indice_transicion <> 0
                                transicion ← celda_transicion(indice_transicion)
                                almacena(letra(indice_transicion,automata),
                                obten_enlace(estado_destino(indice_transicion)), celda)
```

```
                    escribe_en_fichero(celda)
                    indice_transicion ← siguiente_transicion(indice_transicion,
                    automata)
                FIN_MIENTRAS
            FIN_DESDE
        FIN_DESDE
FIN_PROCEDIMIENTO
```

2.2.2 NADFA_Mappings

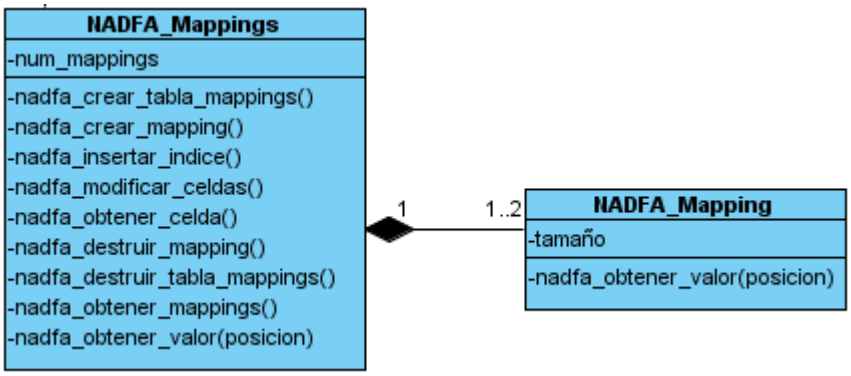

Figura 53. Diagrama de clases parcial de la clase NADFA_Mappings

ATRIBUTOS (NADFA_Mappings)
num_mappings: número de tablas índice.
MÉTODOS (NADFA_Mappings)
nadfa_crear_tabla_mappings(): crea la tabla de índices. **nadfa_crear_mapping():** crea cada uno de los índices. **nadfa_insertar_indice():** inserta una nueva entrada en una tabla *mapping*. **nadfa_modificar_celdas():** modifica los valores de las entradas de una tabla *mapping*. **nadfa_obtener_celda():** obtiene una posición de la tabla *mapping*. **nadfa_destruir_mapping():** elimina una tabla *mapping*. **nadfa_destruir_tabla_mappings():** elimina la tabla de *mappings*. **nadfa_obtener_mappings():** permite acceder a los elementos de las tablas *mapping.* **nadfa_obtener_valor(posicion):** permite obtener el valor almacenado en una posición concreta en las tablas *mapping.*

ATRIBUTOS (NADFA_Mapping)
tamaño: número de entradas del índice (número de formas del diccionario).
MÉTODOS (NADFA_Mapping)
nadfa_obtener_valor(posicion): permite obtener el valor almacenado en una posición concreta en las tablas *mapping.*

Construcción

Cada palabra del autómata tiene su posición correspondiente en cada tabla *mapping,* de modo que cada vez que se lee una palabra se crea una nueva posición en la tabla. En el sistema implementado se utilizan dos tablas *mapping*, una para localizar la posición de la primera información de cada palabra y otra para localizar la posición de la última información de dicha palabra, de modo que si la palabra leída es una palabra repetida no se crea una nueva posición sino que se modifica el índice correspondiente en la tabla *mapping* que almacena la posición de la última información.

Por ejemplo si se lee la palabra "casa" y le corresponde como índice en el autómata el número 1, suponiendo arbitrariamente que en la tabla de información asociada a esa palabra está en la posición 1, en las dos tablas *mapping* se crea una nueva posición y se insertan en el índice 1, recolocando el resto de posiciones de la tabla como puede observarse en la figura:

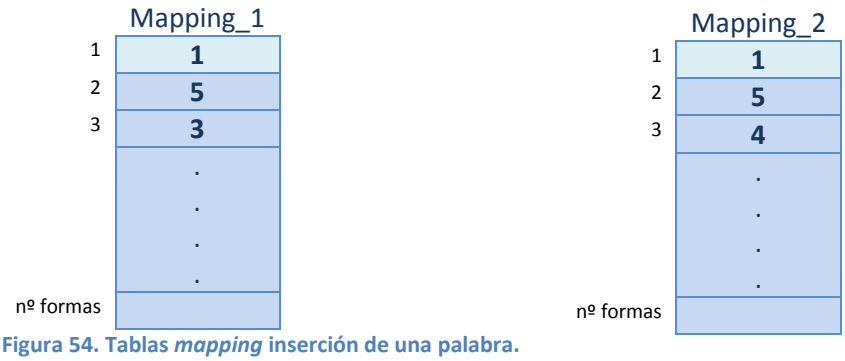

Figura 54. Tablas *mapping* inserción de una palabra.

Si más adelante se vuelve a leer la palabra "casa", no se crea una nueva posición ni se modifica nada en la tabla *mapping* que almacena el índice a la primera información de la palabra, se modifica la posición en la tabla *mapping* a la última información asociada a la palabra, y teniendo en cuenta que las tablas de información no están ordenadas alfabéticamente, puede estar en cualquier posición de la tabla de información, por ejemplo en la posición número 6.

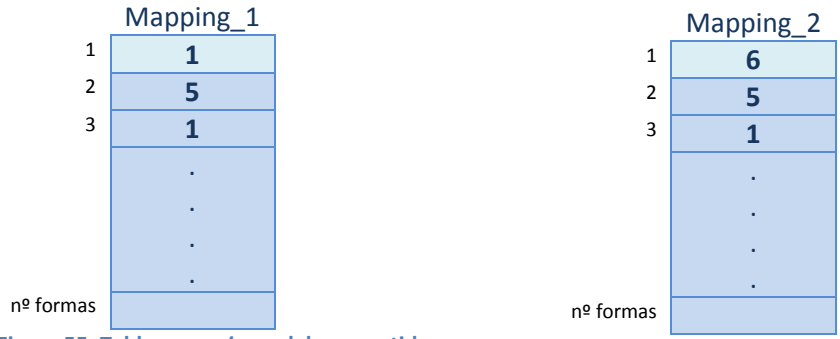

Figura 55. Tablas *mapping* palabra repetida.

Escritura

Para escribir en el fichero binario compilado del diccionario los índices a la información asociada de cada palabra, primero se unifican las tablas *mapping* en un único *array* que, siguiendo el orden alfabético de las palabras del diccionario, almacena los índices de las posiciones de la primera información de cada palabra. Esto está relacionado con la escritura de las tablas de información que, a pesar de no estar ordenadas alfabéticamente en memoria, si se escriben ordenadamente en el fichero binario de salida, de modo que en la unificación de las tablas *mapping* solo sea necesaria una tabla. Para el caso de ejemplo:

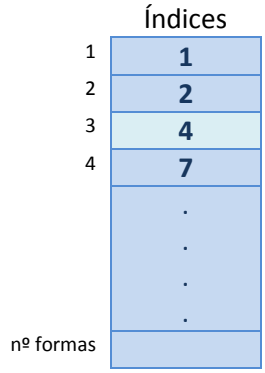

Figura 56. Array con tablas *mapping* unificadas y ordenadas.

El *array* resultado de la unificación y ordenación de las tablas *mapping* es lo que realmente se escribe en el fichero de salida, los índices a las primeras informaciones de cada palabra, por ejemplo, para la palabra que en el autómata se corresponde con la posición 3 en la figura, su primera información en las tablas de información estará en la posición 4, y como para la palabra con la posición 4 será la posición 7, se sabe que la palabra actual tiene 3 celdas de información, por lo que en las tablas de información ocupará las posiciones 4,5 y 6 (en la 7 empieza la información de la siguiente palabra por orden alfabético como se verá en el apartado siguiente).

2.2.3 NADFA_Infos

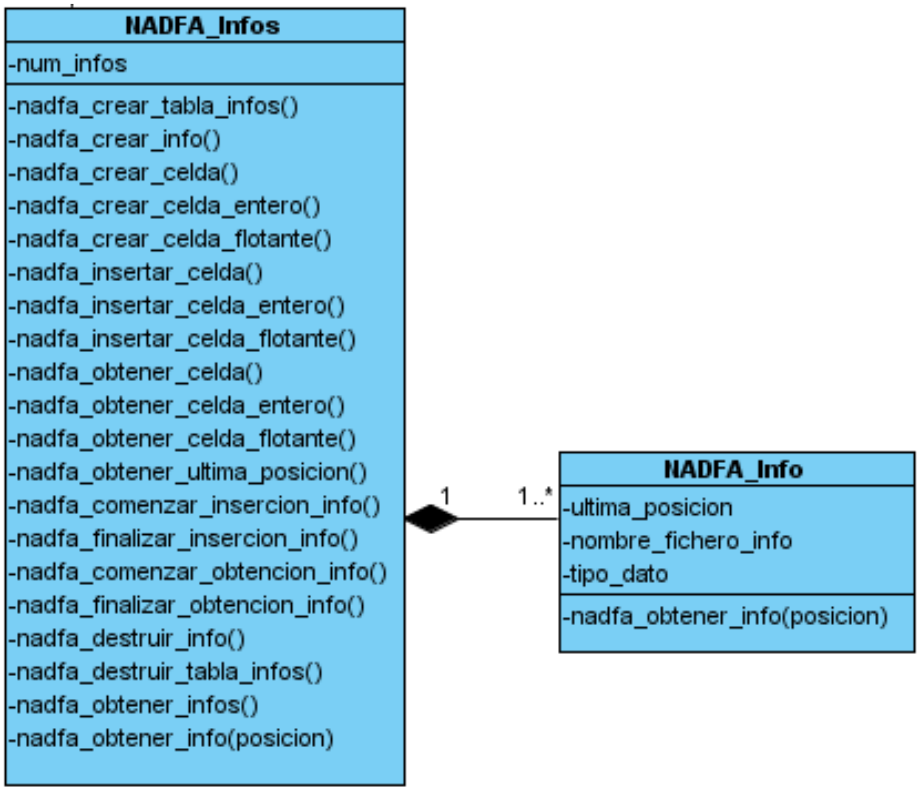

Figura 57. Diagrama de clases de la clase NADFA_Infos

ATRIBUTOS (NADFA_Infos)
num_infos: número de columnas de información.
MÉTODOS (NADFA_Infos)
nadfa_crear_tabla_infos(): crea la tabla de columnas de información.
nadfa_crear_info(): crea cada una de las columnas de información.
nadfa_crear_celda(): crea una casilla de una columna de información, independientemente del tipo de dato que haya en el interior de esa posición.
nadfa_crear_celda_entero(): crea una casilla de una columna de información en la que los datos son de tipo entero.
nadfa_crear_celda_flotante(): crea una casilla de una columna de información en la que los datos son de tipo flotante.
nadfa_insertar_celda(): inserta una casilla en una columna de información, independientemente del tipo de dato que haya en el interior de esa posición.
nadfa_insertar_celda_entero(): inserta una casilla en una columna de información en la que los datos son de tipo entero.
nadfa_insertar_celda_flotante(): inserta una casilla en una columna de información en la que los datos son de tipo flotante.
nadfa_obtener_celda(): obtiene una casilla de una columna de información, independientemente del tipo de dato que haya en el interior de esa posición.
nadfa_obtener_celda_entero(): obtiene una casilla de una columna de información en la que el dato es de tipo entero.
nadfa_obtener_celda_flotante(): obtiene una casilla de una columna de información en la que el dato es de tipo doble.
nadfa_obtener_ultima_posicion(): obtiene la última posición insertada en la tabla de columnas de

información.
nadfa_comenzar_insercion_info(): abre el fichero que contendrá la columna de información correspondiente y crea una casilla a insertar.
nadfa_finalizar_insercion_info(): cierra el fichero que contiene la columna de información correspondiente.
nadfa_comenzar_obtencion_info(): abre el fichero que contiene la columna de información de la que se quiere obtener algún dato.
nadfa_finalizar_obtencion_info(): cierra el fichero del que se ha obtenido información.
nadfa_destruir_info(): elimina una columna de información.
nadfa_destruir_tabla_infos(): elimina la tabla de columnas de información.
nadfa_obtener_infos(): permite acceder a los elementos de las tablas de información.
nadfa_obtener_info(posicion): permite acceder al contenido de una posición en una tabla de información.

ATRIBUTOS (NADFA_Info)
ultima_posicion: última posición insertada la columna de información.
nombre_fichero_info: fichero en el que se almacena la columna de información.
tipo_dato: tipo de dato del contenido de cada posición de la columna de información.
MÉTODOS (NADFA_Info)
nadfa_obtener_info(posicion): permite acceder al contenido de una posición en una tabla de información.

Construcción

En la figura que se muestra a continuación se describe como se almacenan en NADFA las tablas *mapping* y las tablas de información, resaltando a modo de ejemplo los valores para el índice Mapping [2] donde la información asociada sería la de las entradas Info_X[2], Info_X[4] e Info_X[6]. No son posiciones consecutivas porque la información no está ordenada, pero en cada posición se almacena la posición de la siguiente información para la palabra.

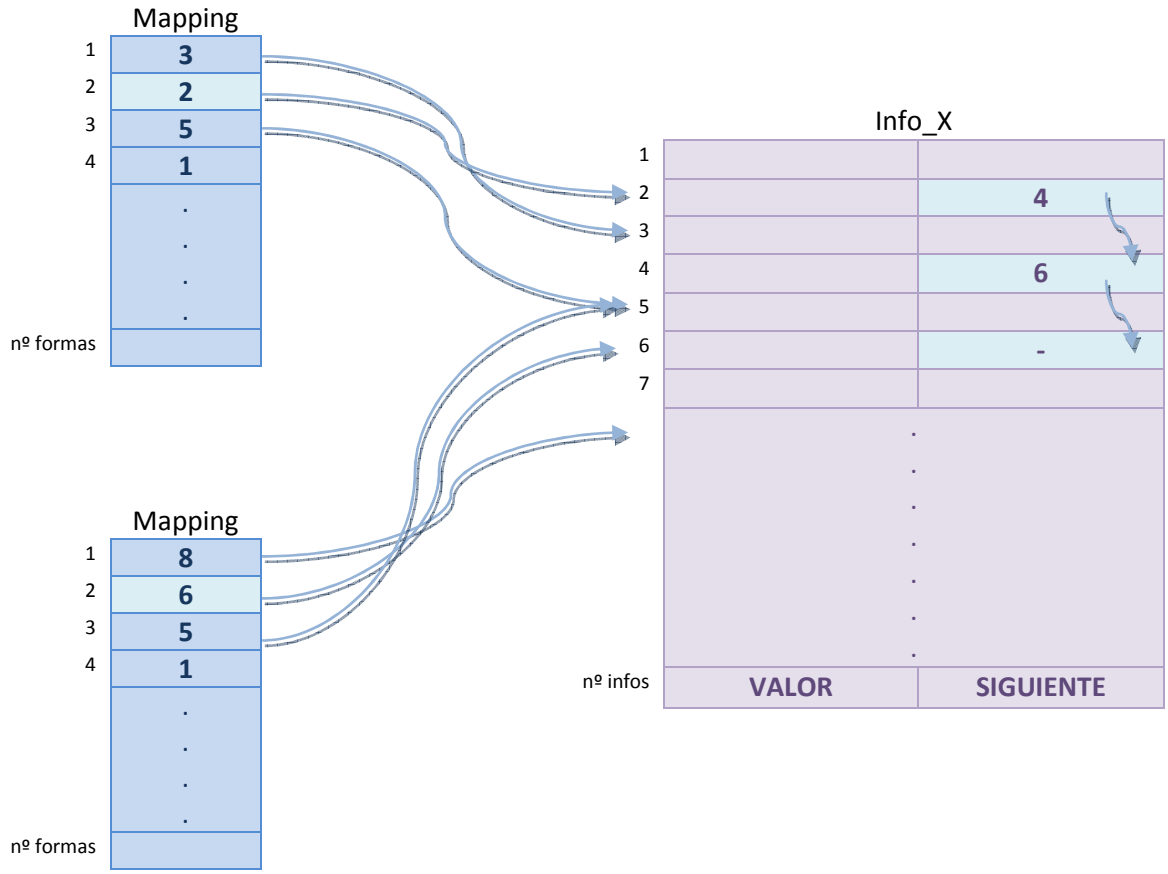

Figura 58. Estructura NADFA.

Cada celda de información se representa mediante una estructura que tiene 2 partes:

Valor: el contenido de la columna de información.

Siguiente: la posición de la siguiente entrada en caso de que la palabra asociada a dicha información tenga más de una aparición en el diccionario.

La información a almacenar de las columnas de información puede ser el dato propiamente dicho o puede ser un índice, que indique la posición donde se almacena el identificador correspondiente a otro autómata previamente compilado.

Para obtener ese valor a almacenar se aplica la función **palabra_a_Índice** sobre el autómata compilado correspondiente, como refleja la siguiente figura:

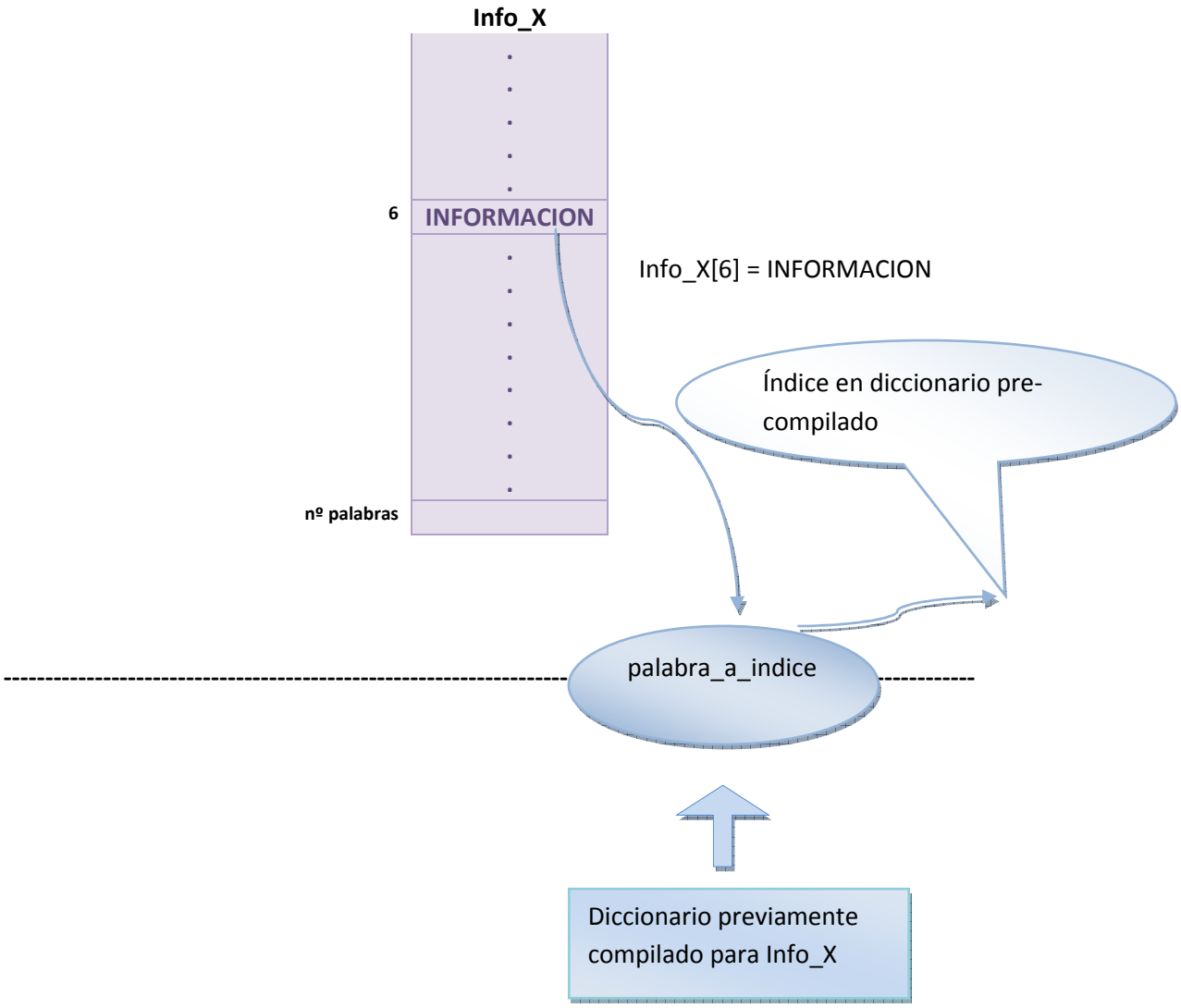

Figura 59. Acceso autómata pre-compilado.

Escritura

Finalmente, el compilador escribirá ordenadamente todas las tablas de información para que las palabras con la misma forma estén indexadas consecutivamente (ya que el diccionario inicial de palabras no está ordenado alfabéticamente), de modo que para una palabra se podrá acceder a todas las entradas que pueda tener en el diccionario (informaciones diferentes para la misma palabra). Este proceso implica reorganizar también las tablas *mapping* que se unifican en una solo *mapping* con una entrada por cada forma (en orden alfabético) como se ha comentado en el apartado anterior. Una vez terminada esta reorganización se escriben

tanto la tabla *mapping* como las de información en el fichero binario de salida, obteniendo así la versión compilada del diccionario. Cada entrada del *mapping* apunta a la primera aparición de cada palabra en las tablas de información, todas las posiciones intermedias entre la entrada actual y la siguiente del *mapping* corresponden al resto de entradas para la misma forma.

Esta es la reordenación y resultado final en el fichero binario que será utilizado por NADFA_BIN para el acceso a los diccionarios. Para la posición número 3 en la tabla *mapping* la información asociada sería la de las entradas:

Info_1[4], Info_1[5], Info_1[6]

Info_2[4], Info_2[5], Info_2[6]

Info_3[4], Info_3[5], Info_3[6]

…

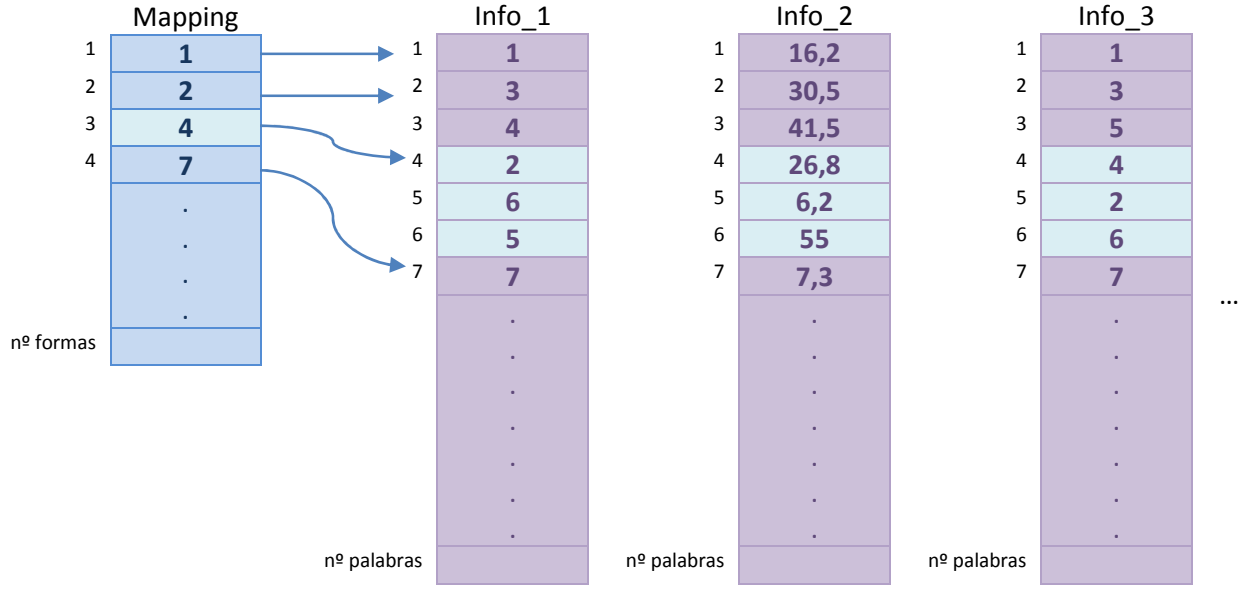

Figura 60. Almacenamiento ordenado de la información del diccionario en fichero binario

2.3 NADFA_BIN

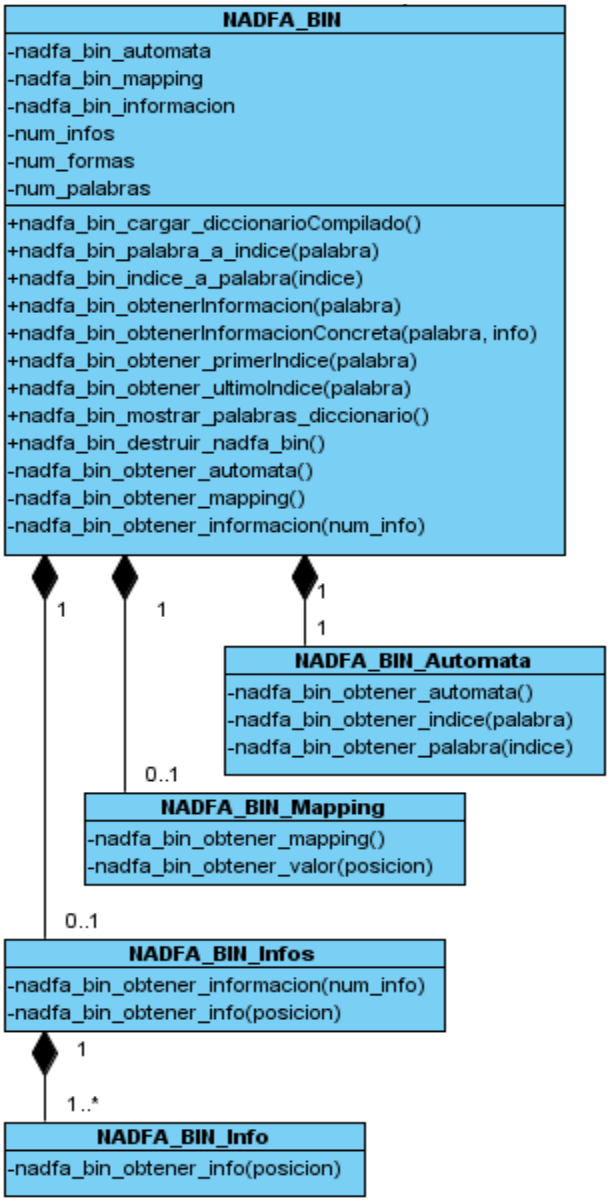

Figura 61. Diagrama de clases parcial de la clase NADFA_BIN

Esta clase es la encargada de controlar el manejo de la librería. Contiene las funciones de acceso al diccionario de palabras comprimido en el fichero binario de salida.

ATRIBUTOS
nadfa_bin_automata: array que almacena el autómata recuperado del fichero binario que contiene la versión comprimida del diccionario.
nadfa_bin_mapping: array que almacena el índice a la información de las palabras recuperado del fichero binario que contiene la versión comprimida del diccionario.
nadfa_bin_informacion: array que almacena la información de las palabras recuperada del fichero

binario que contiene la versión comprimida del autómata.
num_infos: número de columnas de información.
num_formas: número de formas en el diccionario.
num_palabras: número de palabras del diccionario.

MÉTODOS

nadfa_bin_cargar_diccionarioCompilado(): carga en memoria el fichero binario que contiene el diccionario comprimido.
nadfa_bin_palabra_a_indice(palabra): devuelve el índice de una palabra en el diccionario comprimido.
nadfa_bin_indice_a_palabra(indice): devuelve la palabra correspondiente a un índice concreto en el diccionario comprimido.
nadfa_bin_obtenerInformacion(palabra): obtiene toda la información asociada a una palabra del diccionario, esto es, toda la información asociada a todas las entradas con la misma forma para una palabra.
nadfa_bin_obtenerInformacionConcreta(palabra, info): obtiene la información solicitada para la palabra pasada como parámetro.
nadfa_bin_obtener_primerIndice(palabra): obtiene el primer índice en el que aparece la palabra pasada como parámetro.
nadfa_bin_obtener_ultimoIndice(palabra): obtiene el índice en el que aparece la última entrada asociada a la palabra pasada como parámetro.
nadfa_bin_mostrar_palabras_diccionario(): muestra todas las palabras que reconoce el autómata, es decir, todas las palabras del diccionario.
nadfa_bin_destruir_nadfa_bin(): elimina la estructura creada en memoria y que almacena el diccionario comprimido.
nadfa_bin_obtener_automata(): permite acceder a los elementos del autómata.
nadfa_bin_obtener_mapping(): permite acceder a los elementos del índice.
nadfa_bin_obtener_informacion(num_info): permite acceder a la información.

La función **nadfa_bin_cargar_diccionarioCompilado()** permite acceder al fichero binario que contiene la versión compilada del diccionario y cargarla en memoria, esto se lleva a cabo en tres partes, primero se cargan los datos correspondientes al autómata que contiene las palabras del diccionario, luego los datos referidos a los *mappings* de las palabras y finalmente la información propiamente dicha. Como ya se ha comentado, el objeto NADFA_BIN en la implementación del sistema se corresponde con una estructura formada por 3 *arrays*, uno que almacena las celdas que en el fichero binario se corresponden con el autómata compilado, otro que almacena las celdas que en el fichero se corresponden con la tabla de índices *mapping* y otro para las tablas de información.

Autómata: para cargar en memoria el autómata se accede a la celda el fichero binario que contiene la versión compilada del diccionario. La primera posición del fichero contiene lo primero que se había escrito al escribir el autómata, el número de posiciones ocupadas por el mismo, lo que permite saber lo que debe ocupar el *array* que almacena el autómata y reservar memoria para él. Seguidamente se leen las celdas estado seguida cada una por sus correspondientes celdas transición y se almacenan tal cual en el *array* del autómata correspondido con el objeto NADFA_BIN_Automata.

Mapping: la carga en memoria de los datos correspondientes al *mapping* de las palabras se hace inmediatamente después de cargar el autómata. Una vez leídas todas las posiciones que ocupa el autómata en el fichero binario, la siguiente posición del mismo indica el número de posiciones que ocupa la tabla de índices (*mapping*) lo que permite saber hasta dónde hay que

leer en el fichero para cargar dicha tabla, a la vez que cuánto espacio en memoria se debe reservar para la misma.

Seguidamente se leen tantas posiciones como indica el valor obtenido y se almacenan en el *array* correspondiente al objeto NADFA_BIN_Mapping.

Información: A continuación se carga en memoria la información asociada a las palabras del diccionario. Se hace en un *array* que almacena las tablas de información, cada una de las cuales es a su vez un *array* cuyas posiciones son las entradas de información propiamente dichas.

La primera posición que se lee, la siguiente a la última posición correspondiente a la tabla *mapping,* es el número de tablas de información, es decir el número de columnas de información en el diccionario inicial, y seguidamente el tamaño de cada una de las tablas, lo que permite saber las dimensiones del *array* que permite la carga en memoria. Después por cada una de las columnas de información primero se lee el tipo de los datos de esa columna y luego los datos que se cargan.

Una vez cargada en memoria la versión compilada del diccionario se puede acceder a su contenido mediante el manejo de las funciones de la librería. A continuación se muestra el pseudocódigo de las principales funciones, que permiten el acceso a los datos del diccionario compilado:

El pseudocódigo de la función **nadfa_bin_palabra_a_indice(palabra, automata)** es el mismo que el de la función **nadfa_palabra_a_indice(palabra, automata)** mostrado en el apartado anterior. La diferencia entre las dos funciones está en la implementación ya que nadf_palabra_a_indice actúa sobre el autómata inicial generado en memoria antes de ser compilado, y nadfa_bin_palabra_a_indice lo hace sobre el autómata ya compilado y recuperado del fichero binario donde está guardado, por lo que para poder acceder a los datos del autómata debe deshacer las operaciones realizadas para comprimir estados y transiciones a nivel de bit. Lo mismo ocurre entre las funciones **nadfa_bin_indice_a_palabra(indice, automata) y nadfa_indice_a_palabra(indice, automata).**

Si antes se empleaba el operador de desplazamiento hacia la izquierda (<<) para mover los 8 bits de menor peso que almacenan el número de transiciones salientes de un estado hacia la izquierda, ahora para recuperarlas se emplea el operador de desplazamiento hacia la derecha (>>) para volver a su posición original.
número de transiciones = celda_estado >> 24

En cambio para recuperar el peso de dicho estado, que antes se almacenaba en los 24 bits de menor peso (a la derecha) mediante el operador de suma, ahora se recupera mediante el operador AND[7] (&) .
peso = celda_estado & 0X00FFFFFF

El número 0X00FFFFFF en formato hexadecimal representa 32 bits de los cuales los 8 bits de mayor peso (a la izquierda) son ceros en formato binario y los 24 bits restantes (a la derecha) son unos, que coinciden con los 24 bits que en una celda estado están reservados para el peso.

[7] Operador conjunción donde si todos los operandos son **1** devuelve **1**; si hay alguno que sea **0** devuelve **0**.

De este modo mediante el operador AND solo es posible recuperar el valor del peso, de esos 24 bits, a modo de ejemplo:

Si el peso de un estado es 3 en formato decimal, en formato binario y tal como se almacena en una celda estado es: 00000000 00000000 00000000 00000011

El valor 0X00FFFFFF en formato hexadecimal, en formato binario es: 00000000 11111111 11111111 11111111

De modo que al aplicar el operador AND sólo hay la posibilidad de recuperar el valor de los 24 bits reservados para el peso, aquellos que sean un uno permanecerán a uno:

00000000 00000000 00000000 00000011
&
00000000 11111111 11111111 11111111

00000000 00000000 00000000 00000011

Lo mismo ocurre con las celdas transición. Si antes se empleaba el operador de desplazamiento hacia la izquierda (<<) para mover los 8 bits de menor peso que almacenan la letra de la transición hacia la izquierda, ahora para recuperarla se emplea el operador de desplazamiento hacia la derecha (>>) para volver a su posición original.
letra = celda_transición >> 24

Para recuperar la posición del estado al que llega dicha transición, que antes se almacenaba en los 24 bits de menor peso (a la derecha) mediante el operador de suma, ahora se recupera también mediante el operador AND (&).
posición del estado al que llega = celda_transición & 0X00FFFFFF

A continuación se muestra el pseudocódigo del resto de funciones de la librería que permiten el acceso al diccionario de palabras, para las que se han empleado llamadas a funciones y métodos con nombres auto-explicativos.

En la función **nadfa_bin_obtener_informacion** se obtiene toda la información almacenada en el diccionario para una palabra. En el pseudocódigo, se emplea la función **necesario_convertir** que permite saber si la información almacenada es realmente el valor a mostrar o es un índice. Si es un índice es necesario su conversión (generalmente mediante la función nadfa_bin_indice_a_palabra) aplicada al autómata previamente compilado correspondiente, que se recupera mediante la función **obten_automata_precompilado**.

El pseudocódigo para la función **nadfa_bin_obtener_informacionConcreta** es muy similar, varía en que antes de mostrar la información se debe localizar cual es el campo de información que se desea obtener para mostrar solamente ese, en lugar de toda la información almacenada para esa palabra.

```
FUNCION nadfa_bin_obtener_informacion(palabra, automata, mapping, informacion)
INICIO
    resultado ← ""
    indice_mapping ← nadfa_bin_palabra_a_indice(palabra, automata)
    SI indice_mapping = 0
```

```
            ESCRIBIR ("LA PALABRA NO ESTA EN EL DICCIONARIO")
        FIN_SI
        SI indice_mapping < num_formas
            num_entradas ← obten_valor(mapping, indice_mapping + 1) -
            obtener_valor(mapping, indice_mapping)
        SINO
            num_entradas ← 1
        FIN_SI
        indice ← obten_valor(mapping, indice_mapping)
        DESDE indice_palabra ← 0 HASTA num_entradas
            DESDE indice_info ← 0 HASTA num_infos
                concatenar(resultado,nombre_info(indice_info))
                SI tipo_info(indice_info) = entero
                    SI necesario_convertir(indice_info)
                        automata_precompilado ←
                        obten_automata_precompilado (indice_info)
                        concatenar(resultado,
                        obten_entero(nadfa_bin_indice_a_palabra
                        (informacion[indice_info][indice]),
                        automata_precompilado))
                    SINO
                        concatenar(resultado,
                        obten_entero(informacion[indice_info][indice]))
                    FIN_SI
                SINO
                    concatenar (resultado, obten_doble (informacion
                    [indice_info][indice]))
                FIN_SI
            FIN_DESDE
            indice ← indice + 1
        FIN_DESDE
        DEVOLVER resultado
FIN_FUNCION
```

```
FUNCION
nadfa_bin_obtener_informacionConcreta(palabra,id_info,automata,mapping,informacion)
INICIO
        resultado ← “”
        indice_mapping ← nadfa_bin_palabra_a_indice(palabra, automata)
        SI indice_mapping = 0
                ESCRIBIR ("LA PALABRA NO ESTA EN EL DICCIONARIO")
        FIN_SI
        SI indice_mapping < num_formas
                num_entradas ← obten_valor(mapping, indice_mapping + 1) -
                obtener_valor(mapping, indice_mapping)
        SINO
                num_entradas ← 1
        FIN_SI
        indice ← obten_valor(mapping, indice_mapping)
        DESDE indice_infos ← 0 HASTA num_infos
                SI obten_nombre_info(indice_infos) = obten_nombre_info(id_info)
                        indice_info ← indice_infos
                FIN_SI
        FIN_DESDE
        DESDE indice_palabra ← 0 HASTA num_entradas
                SI tipo_info(indice_info) = entero
                        SI necesario_convertir(indice_info)
                                automata_precompilado ← obten_automata_precompilado
                                (indice_info)
                                concatenar(resultado,obten_entero(nadfa_bin_indice_a_palabra
                                (informacion[indice_info][indice]), automata_precompilado))
                        SINO
                                concatenar(resultado,
                                obten_entero(informacion[indice_info][indice]))
                        FIN_SI
                SINO
                        concatenar (resultado, obten_doble (informacion [indice_info][indice]))
                FIN_SI
                indice ← indice + 1
        FIN_DESDE
        DEVOLVER resultado
FIN_FUNCION
```

```
FUNCION nadfa_bin_dar_primerIndice(palabra, automata, mapping, informacion)
INICIO
        indice_mapping ← nadfa_bin_palabra_a_indice (palabra, automata)
        SI indice_mapping = 0
                ESCRIBIR ("LA PALABRA NO ESTA EN EL DICCIONARIO")
        FIN_SI
        indice ← obtener_valor(mapping, indice_mapping)
        DEVOLVER indice
FIN_FUNCION
```

Por último el pseudocódigo para la función nadfa_bin_dar_ultimoIndice:

```
FUNCION nadfa_bin_dar_ultimoIndice(palabra, automata, mapping, informacion)
INICIO
        indice_mapping ← nadfa_bin_word_to_index (palabra, automata)
        SI indice_mapping = 0
                ESCRIBIR ("LA PALABRA NO ESTA EN EL DICCIONARIO")
        FIN_SI
        num_formas ← obtén_num_formas(automata)
        SI indice_mapping = num_formas
                indice ← obten_valor(mapping, indice_mapping)
        SINO
                palabra_siguiente ← obten_palabra_siguiente (palabra, automata)
                indice_mapping_palabra_siguiente ← nadfa_bin_palabra_a_indice
                (palabra_siguiente, autómata)
                indice ← obten_valor(mapping, palabra_siguiente) - 1
        FIN_SI
        DEVOLVER indice
FIN_FUNCION
```

2.4 Entrada/Salida

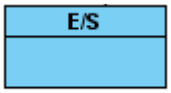

Figura 62. Diagrama de clases parcial de la clase E/S

Esta clase engloba a todos los ficheros que intervienen en la ejecución del sistema, es decir, el fichero de configuración y todos los ficheros de entrada y salida.

FICHERO DE CONFIGURACIÓN

El fichero de configuración permite obtener los datos de configuración necesarios para llevar a cabo la compilación de un diccionario y que varían en función de las necesidades del usuario.

Permite localizar el fichero que contiene el diccionario de palabras junto con la información de cada una de ellas, y saber dónde se almacenará su versión compilada, además del número de tablas *mapping* para indexar la información. Habitualmente se utilizarán 2 tablas, una para localizar la primera entrada de cada palabra en el diccionario y otra para localizar la última. Para el caso del fichero de configuración de un autómata previamente compilado este número será 0 debido a que las palabras no tienen información asociada.

Este fichero además incluye por cada columna de información el nombre de dicha columna, la función a aplicar para cada entrada de esa columna en el autómata correspondiente y el número identificador del autómata precompilado (en caso de tener alguno asociado), el tipo de datos de cada casilla en cada una de las columnas de información y al que debe aplicársele la función anteriormente indicada (en caso de ser necesario) y finalmente el fichero en el que se almacenan cada columna de información por separado.

Por último el fichero identifica cada número de autómata previamente compilado con el fichero binario en el que está comprimido, reservando el “1” para el autómata principal (la información asociada puede ser un índice del propio autómata) y “” si no tiene ningún autómata asociado.

Se puede ver un ejemplo de la estructura del fichero de configuración más adelante en el apartado de Implementación.

3. DIAGRAMAS DE DISEÑO PARA LOS CASOS DE USO DEL SISTEMA

A continuación se muestran los diagramas de secuencia, colaboración y actividades por cada caso de uso, que en el diseño analizan con mayor detalle cada escenario estudiado en el análisis.

3.1. CASO DE USO: COMPILACIÓN DICCIONARIO

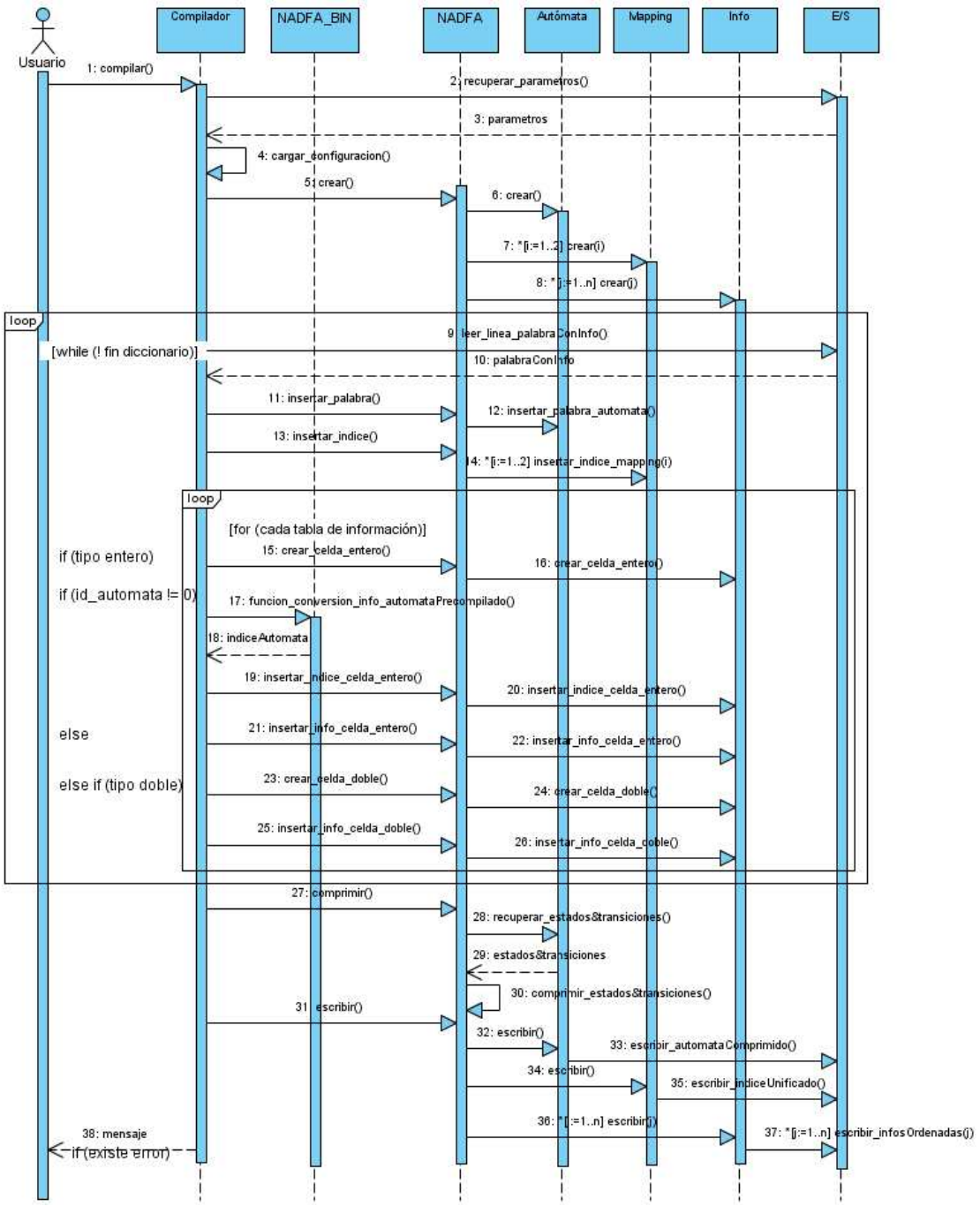

Figura 63. Diagrama de secuencia del caso de uso Compilación Diccionario (Diseño).

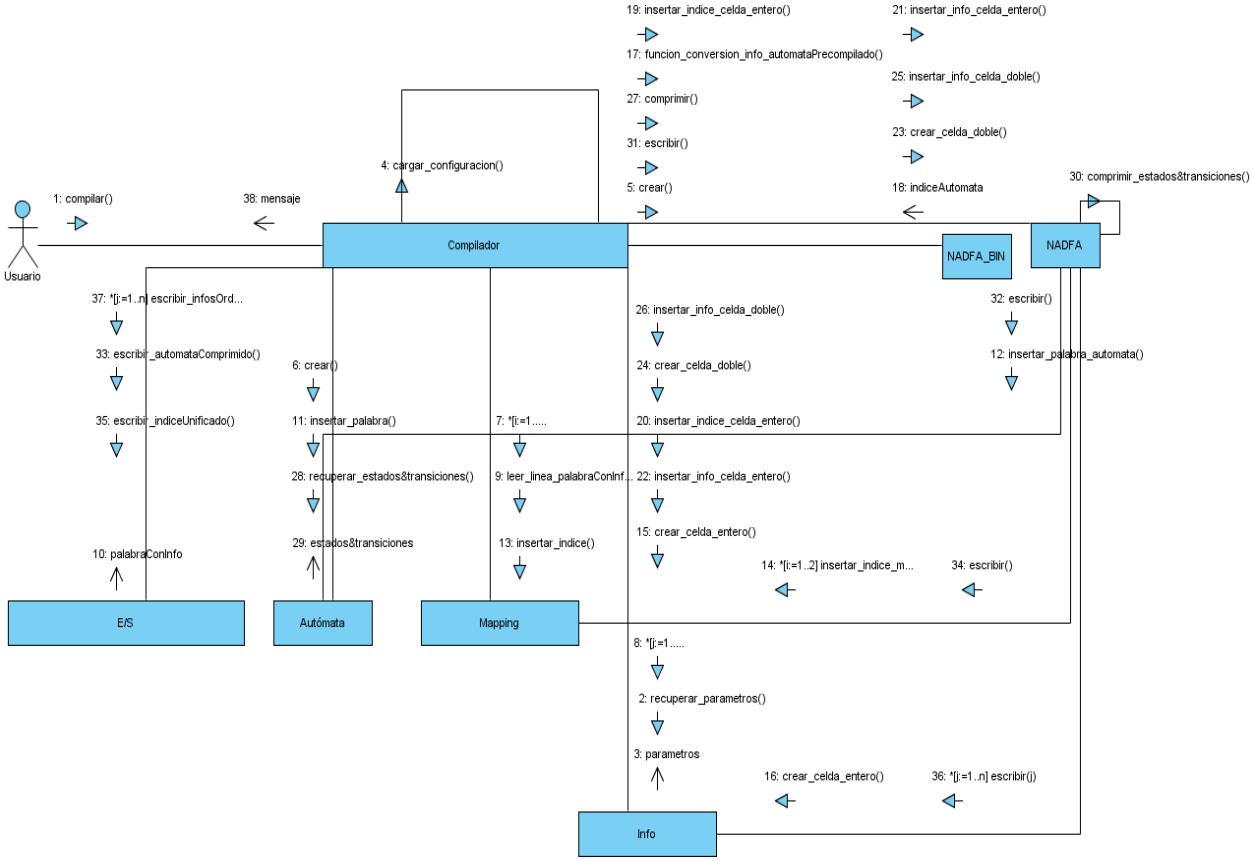

Figura 64. Diagrama de colaboración del caso de uso Compilación Diccionario (Diseño).

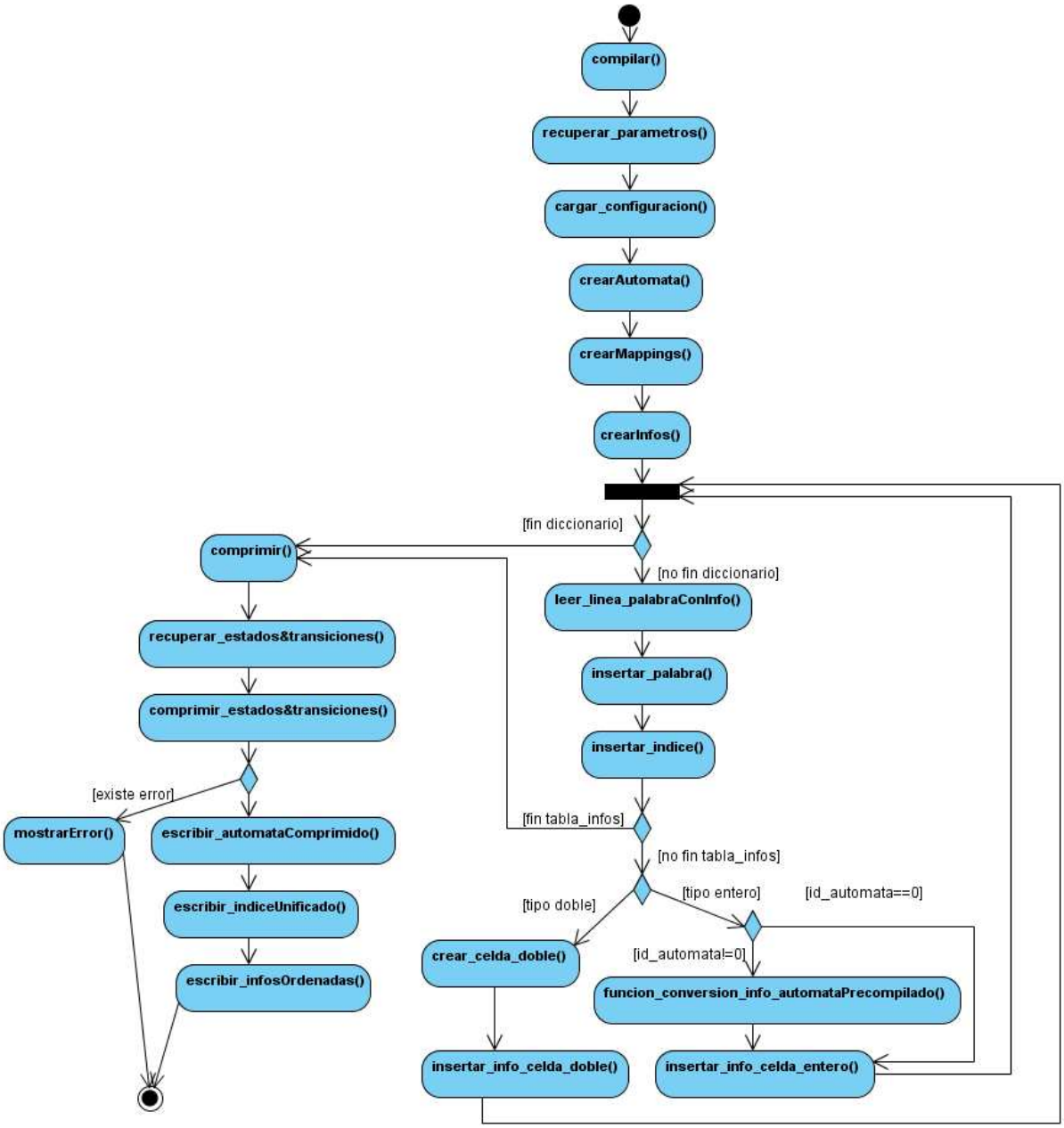

Figura 65. Diagrama de actividades del caso de uso Compilación Diccionario (Diseño)

3.2. CASO DE USO: GESTIÓN DICCIONARIO COMPILADO

3.2.1 ESCENARIO: CARGAR DICCIONARIO COMPILADO EN MEMORIA

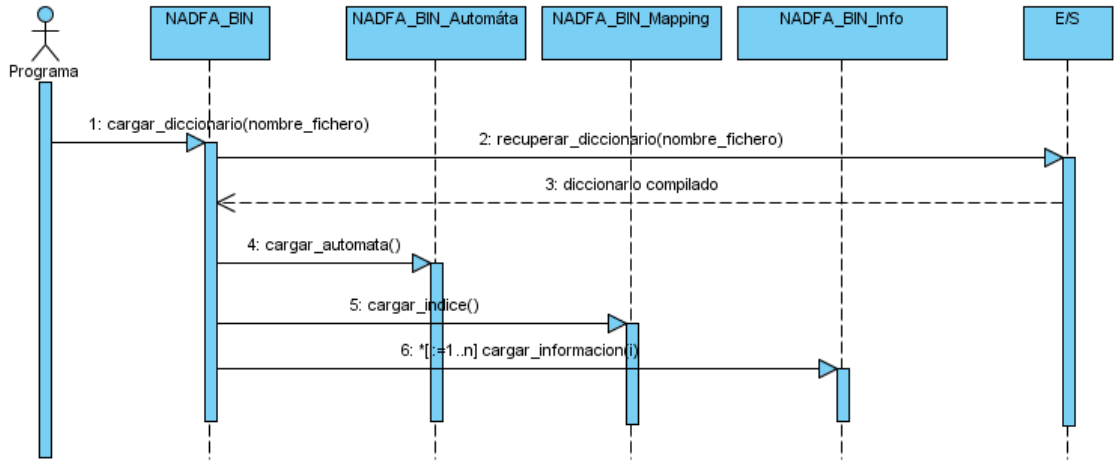

Figura 66. Diagrama de secuencia del caso de uso Cargar diccionario compilado en memoria (Diseño).

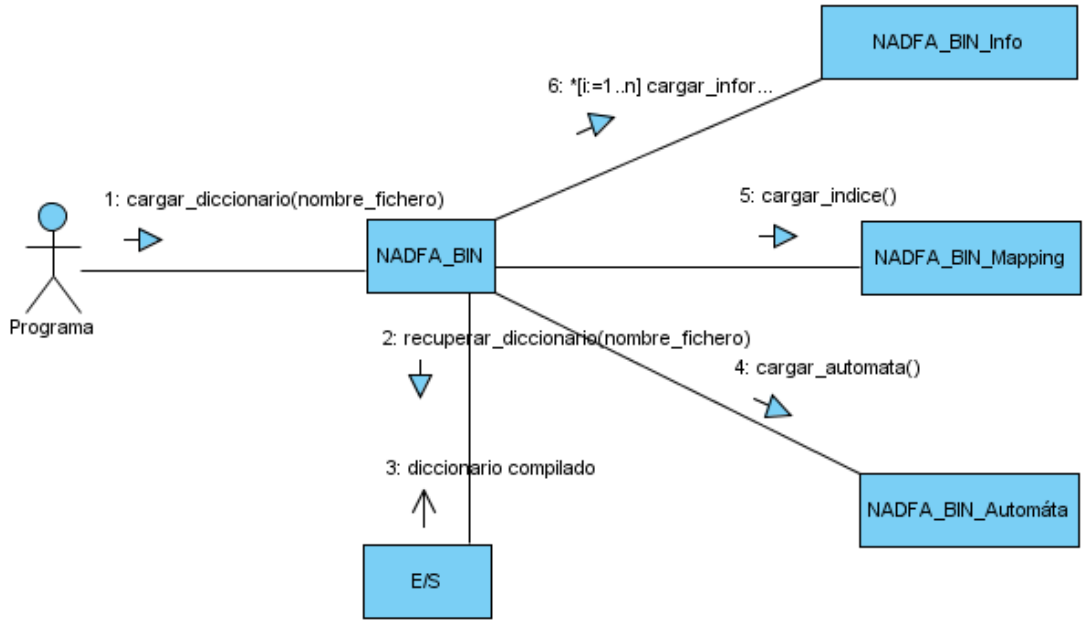

Figura 67. Diagrama de colaboración del caso de uso Cargar diccionario compilado en memoria (Diseño).

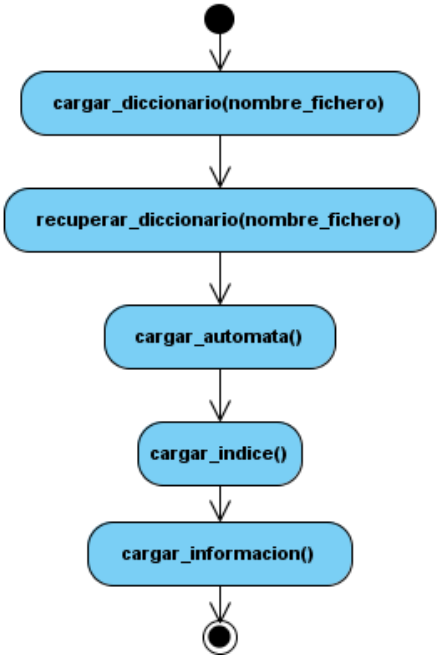

Figura 68. Diagrama de actividades del caso de uso Cargar diccionario compilado en memoria (Diseño).

3.2.2 ESCENARIO: CONVERSIÓN PALABRA A ÍNDICE

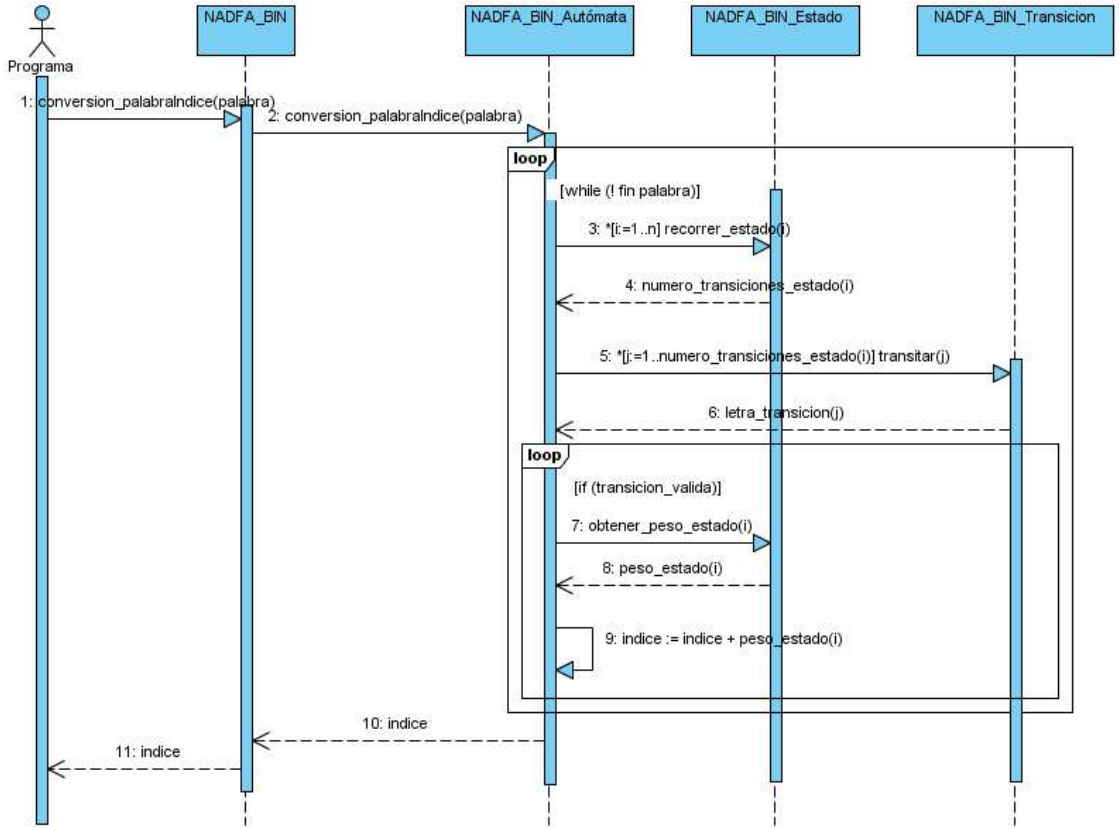

Figura 69. Diagrama de secuencia del caso de uso Conversión Palabra a Índice (Diseño).

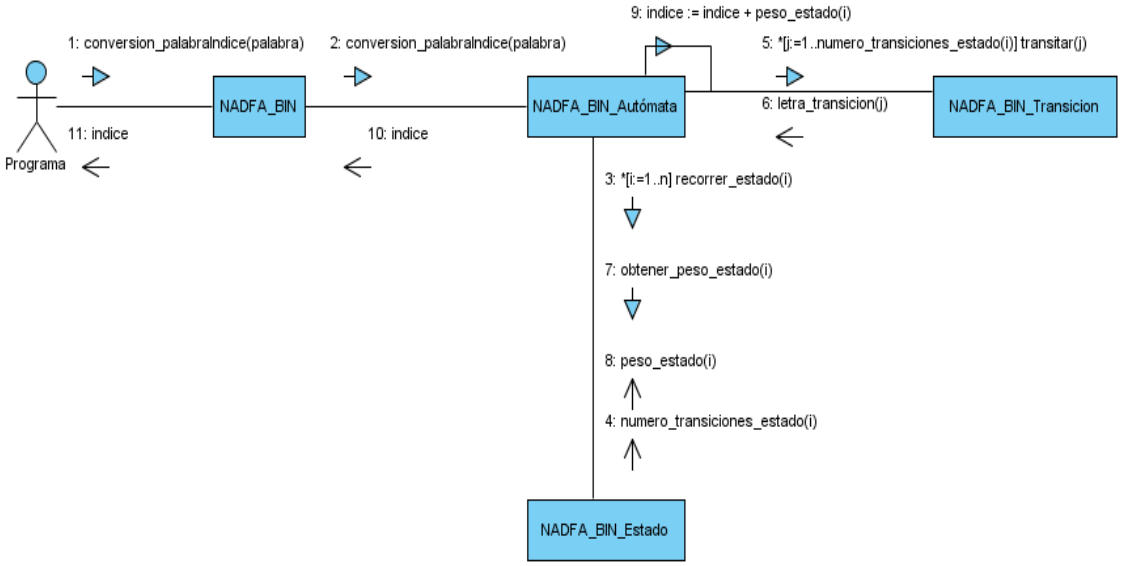

Figura 70. Diagrama de colaboración del caso de uso Conversión Palabra a Índice (Diseño).

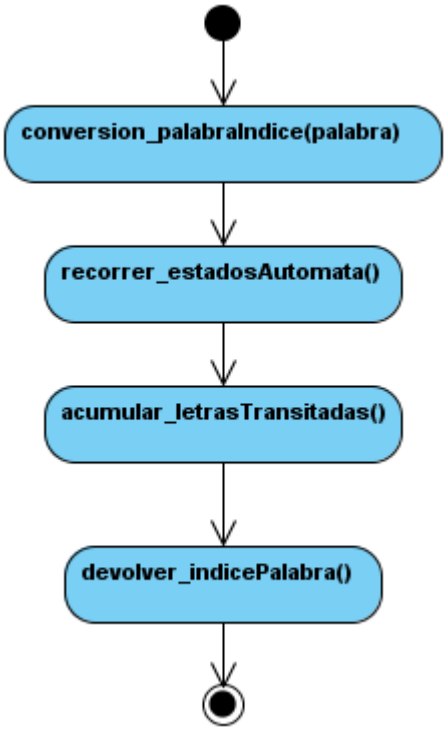

Figura 71. Diagrama de actividades del caso de uso Conversión Palabra a Índice (Diseño)

3.2.3 ESCENARIO: CONVERSIÓN ÍNDICE A PALABRA

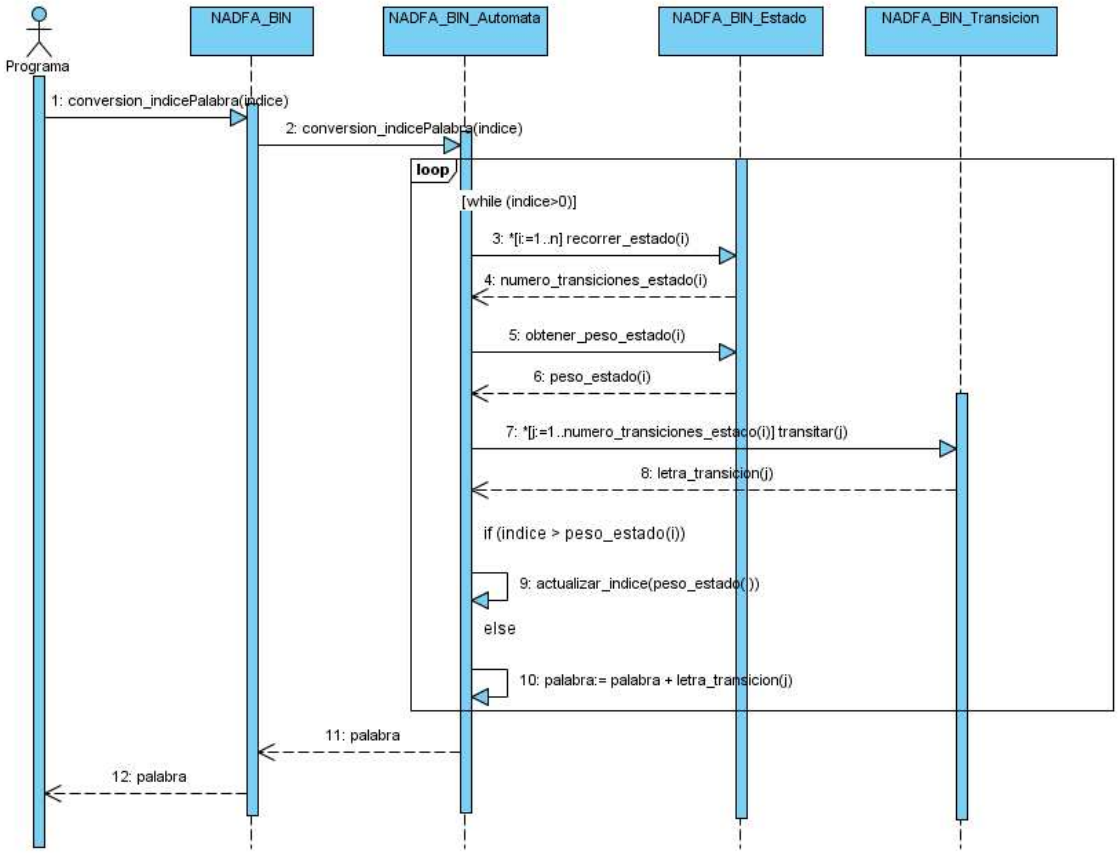

Figura 72. Diagrama de secuencia del caso de uso Conversión Índice a Palabra (Diseño).

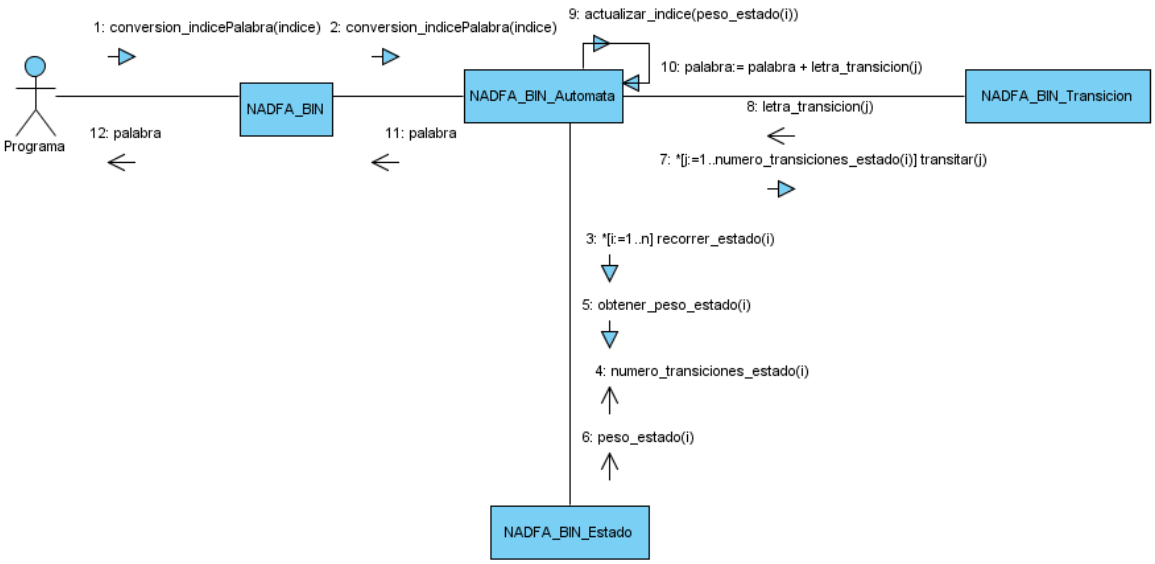

Figura 73. Diagrama de colaboración del caso de uso Conversión Índice a Palabra (Diseño).

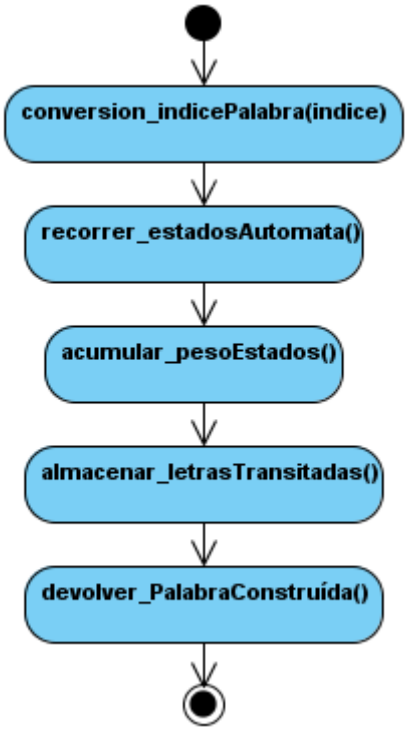

Figura 74. Diagrama de actividades del caso de uso Conversión Índice a Palabra (Diseño)

3.2.4 ESCENARIO: ACCEDER INFORMACIÓN COMPLETA DE PALABRA

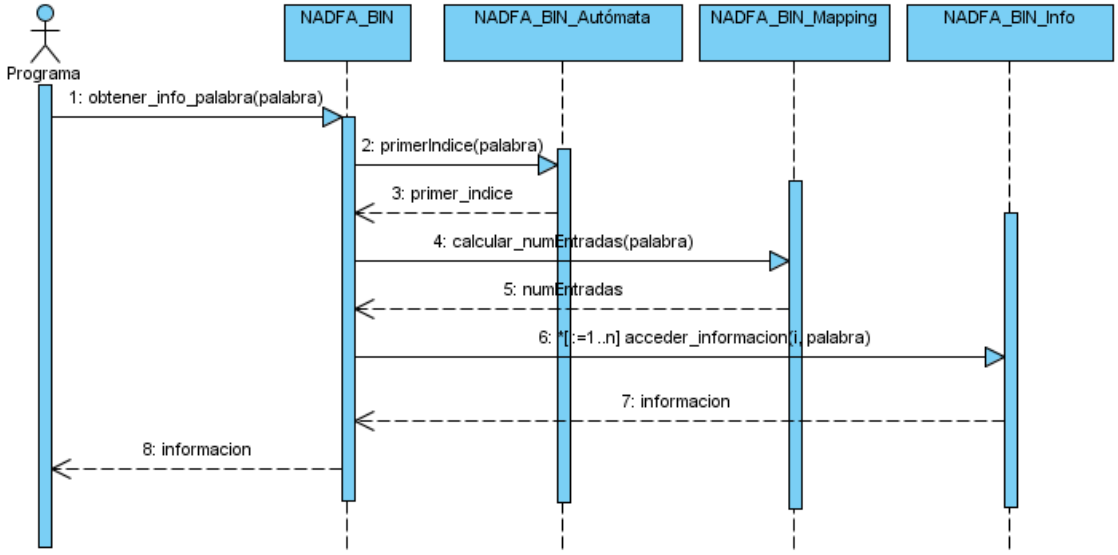

Figura 75. Diagrama de secuencia del caso de uso Acceder Información Completa de Palabra (Diseño).

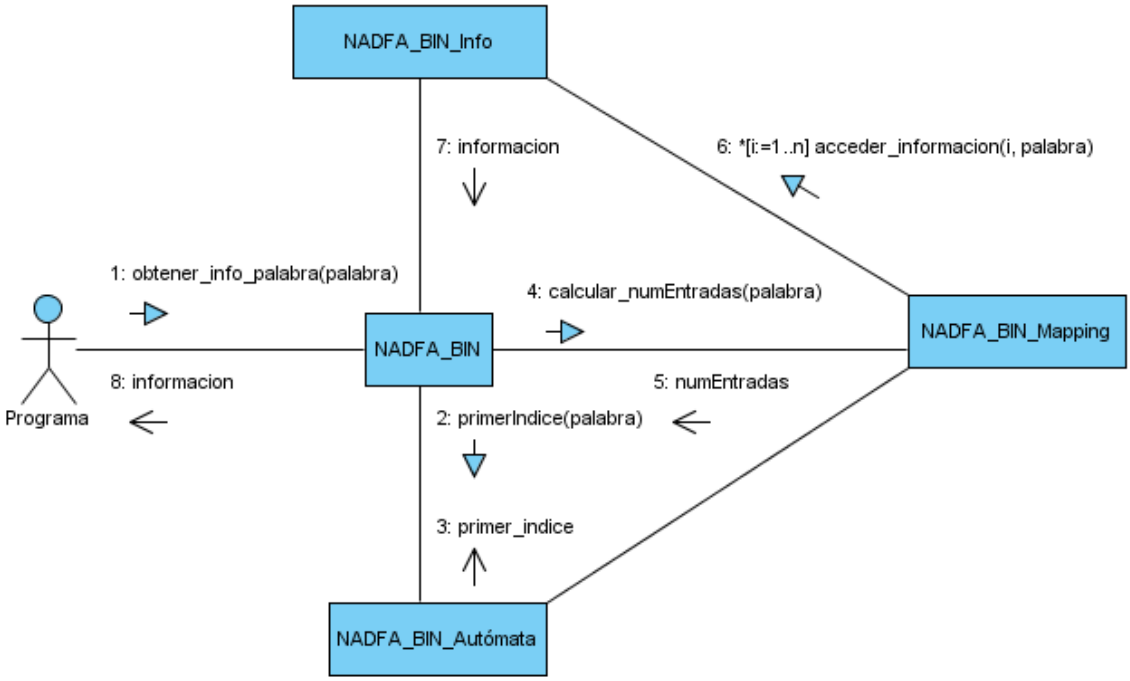

Figura 76. Diagrama de colaboración del caso de uso Acceder Información completa de Palabra (Diseño).

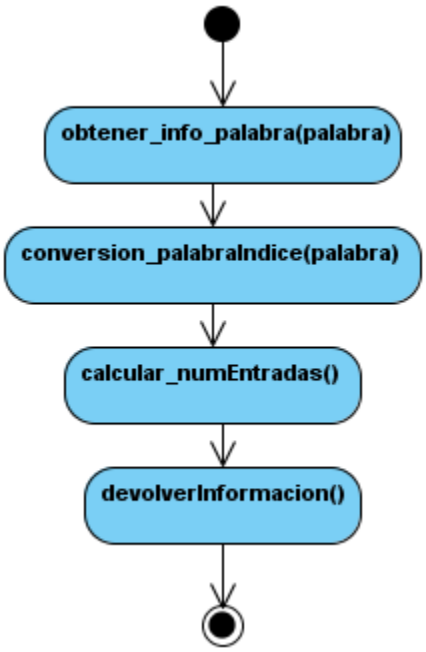

Figura 77. Diagrama de actividades del caso de uso Acceder Información completa de Palabra (Diseño)

3.2.5 ESCENARIO: ACCEDER INFORMACIÓN CONCRETA DE PALABRA

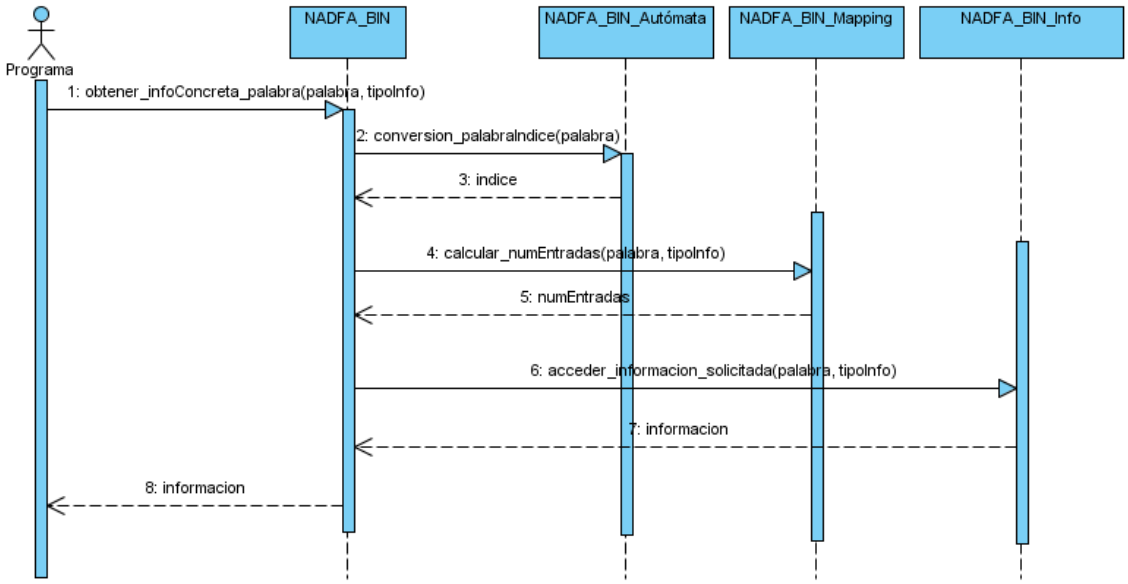

Figura 78. Diagrama de secuencia del caso de uso Acceder Información Concreta de Palabra (Diseño).

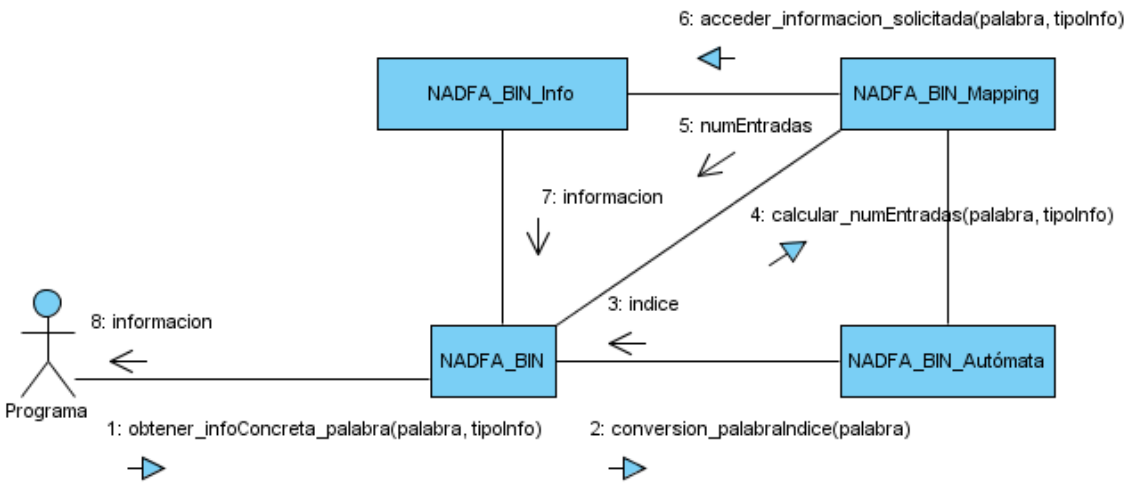

Figura 79. Diagrama de colaboración del caso de uso Acceder Información Concreta de Palabra (Diseño).

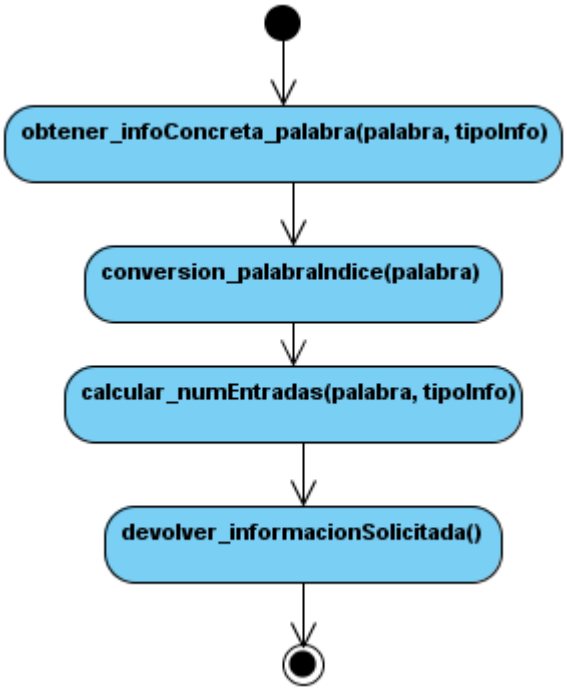

Figura 80. Diagrama de actividades del caso de uso Acceder Información Concreta de Palabra (Diseño)

3.2.6 ESCENARIO: ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA

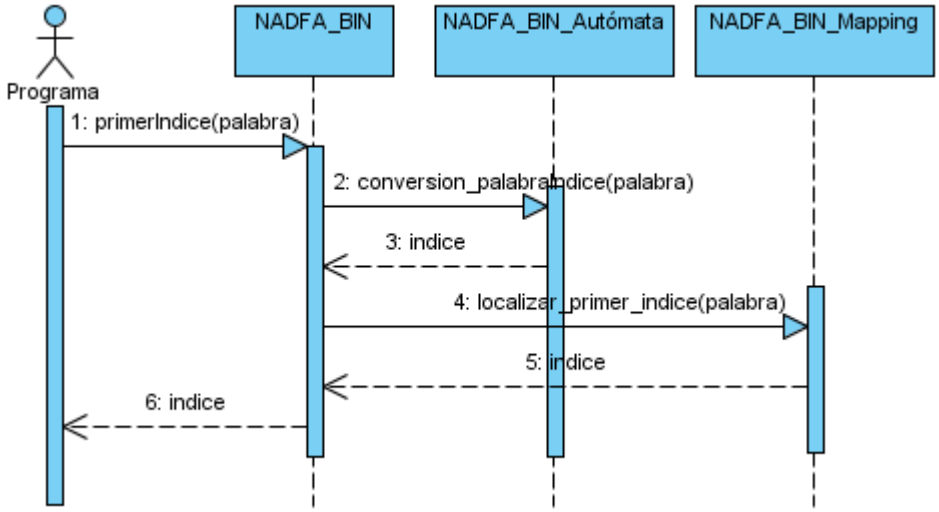

Figura 81. Diagrama de secuencia del caso de uso Índice Primera información de una Palabra (Diseño).

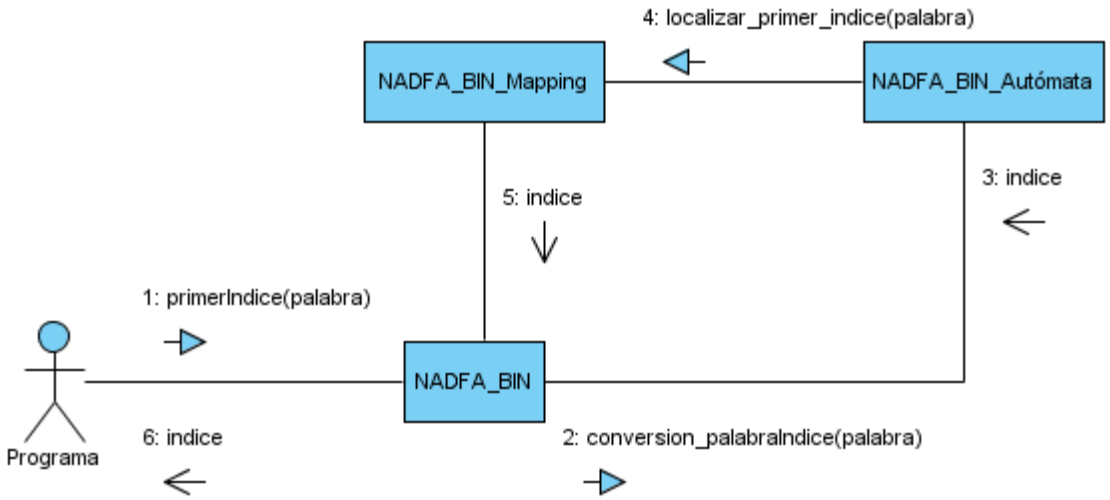

Figura 82. Diagrama de colaboración del caso de uso Índice Primera información de una Palabra (Diseño).

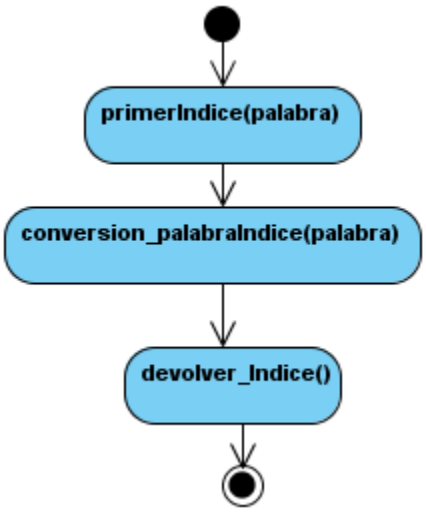

Figura 83. Diagrama de actividades del caso de uso Índice Primera información por Palabra (Diseño)

3.2.7 ESCENARIO: ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA

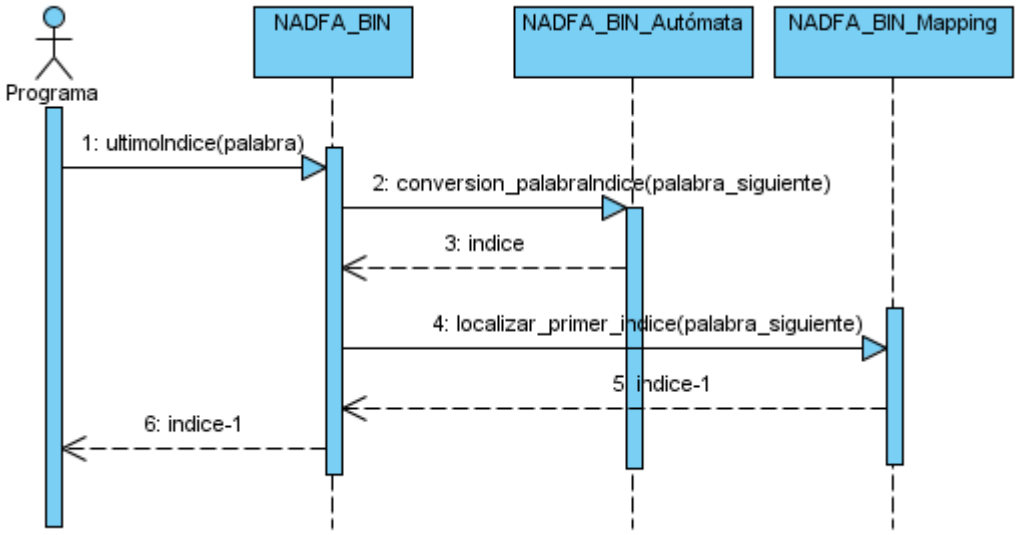

Figura 84. Diagrama de secuencia del caso de uso Índice Última información de una Palabra (Diseño).

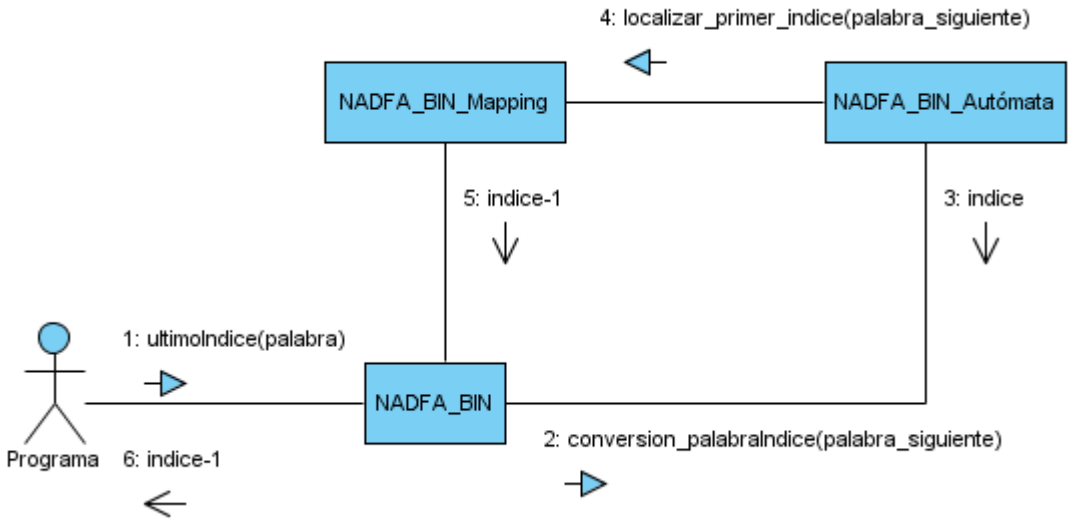

Figura 85. Diagrama de colaboración del caso de uso Índice Última información de una Palabra (Diseño).

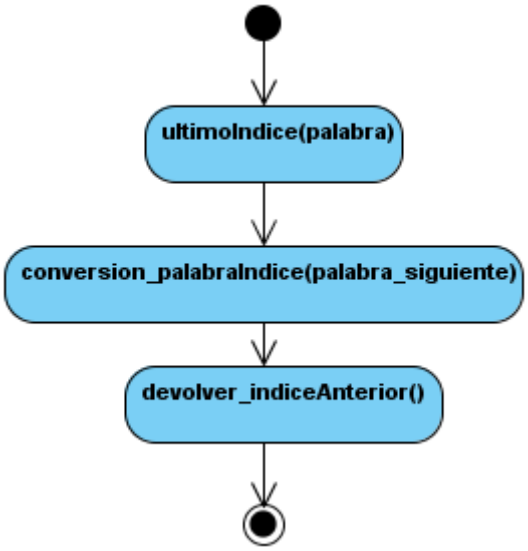

Figura 86. Diagrama de actividades del caso de uso Índice Última información de una Palabra (Diseño)

3.2.8 ESCENARIO: LIBERAR RECURSOS

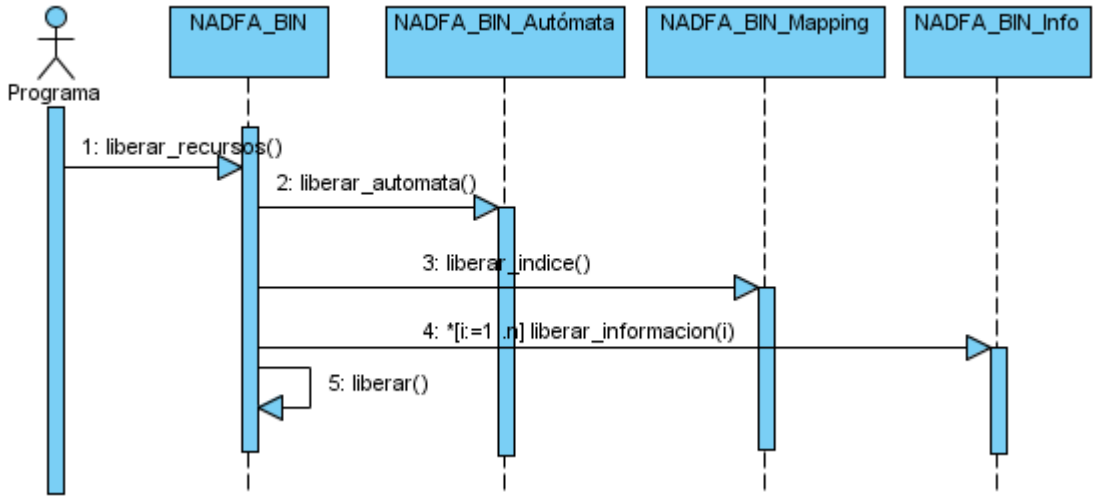

Figura 87. Diagrama de secuencia del caso de uso Liberar recursos (Diseño).

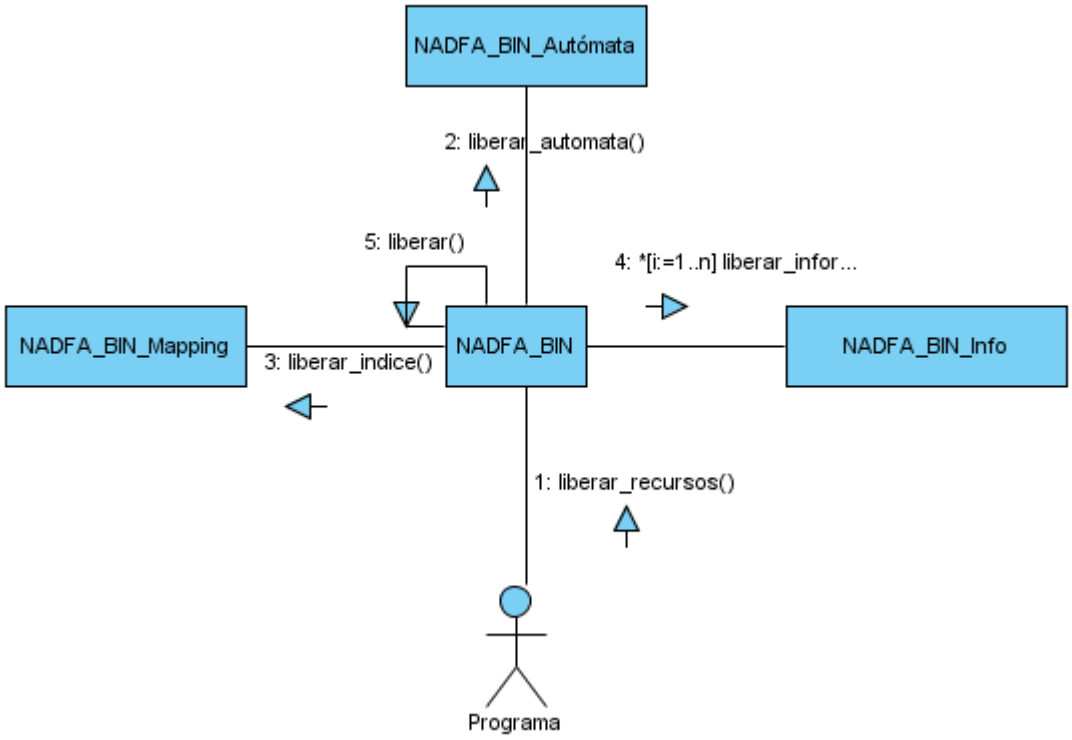

Figura 88. Diagrama de colaboración del caso de uso Liberar recursos (Diseño).

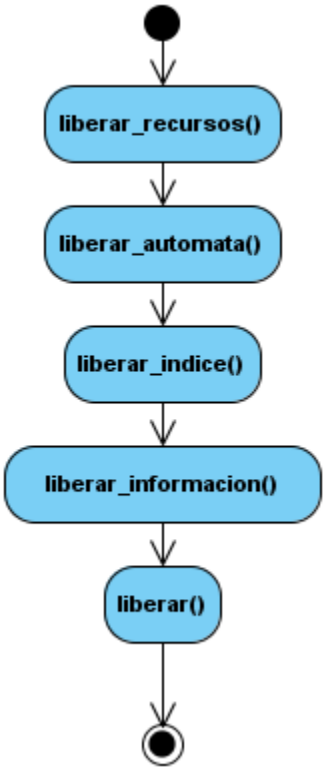

Figura 89. Diagrama de actividades del caso de uso Liberar recursos (Diseño).

IMPLEMENTACIÓN

En esta fase se ponen aplican y ponen en funcionamiento los resultados obtenidos en las fases de análisis y diseño.

Este apartado resulta de interés para comprender, desde una óptica de bajo nivel, el funcionamiento interno del sistema desarrollado. Además se incluye el diagrama de componentes y la estructura del fichero XML de configuración.

GLib (13) es una librería de propósito general que se usa para implementar muchas funciones no gráficas.

Uno de los mayores beneficios de usar GLib es que provee una interfaz de plataforma independiente que permite que el código pueda ser usado en diferentes sistemas operativos, uno de los objetivos a conseguir con este proyecto.

Otro aspecto de GLib es la amplia gama de tipo de datos que deja disponible al desarrollador.

Otra librería importante en el desarrollo de este sistema es Libxml (14), una biblioteca de software para analizar los documentos XML (15). Está escrito en C, y proporciona enlaces a C, C++, XSH, C#, Python, Kylix/Delphi .... Fue originalmente desarrollado para el proyecto GNOME, pero puede ser utilizado fuera de ella.

En Libxml el código es altamente portable, ya que sólo depende de la biblioteca ANSI C y es software libre bajo la licencia MIT. Es muy portable capaz de trabajar sin problemas en una gran variedad de sistemas (Linux, Unix, Windows, Cygwin, MacOS, MacOS X, RISC OS, OS / 2, VMS, QNX, MVS, ...).

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

En el interior del CD-ROM entregado, dentro de la carpeta software se encuentran los ficheros con el código implementado:

Carpeta "libnadfa":
Carpeta "src":

- **Carpeta "testdata":** ficheros necesarios para ejecutar pruebas.
- **"libnadfa.c":** contiene las funciones de manejo de la estructura NADFA.
- **"libnadfa.h":** fichero de cabecera de libnadfa.c.
- **"libnadfa_bin.c":** contiene las funciones de manejo de la estructura LIBNADFA_BIN.
- **"libnadfa_bin.h":** fichero de cabecera de libnadfa_bin.c.
- **"libnadfa_bin_test.c":** programa de pruebas.
- **"libnadfa_test.c":** programa de pruebas.

- **"libnadfac.c":** contiene las funciones empleadas en el proceso de compilación por nadfac.c.
- **"libnadfac.h":** fichero de cabecera de libnadfac.c.
- **"nadfac.c":** programa que lleva a cabo la compilación del diccionario de partida, mediante el manejo de la estructura NADFA.
- **"testing.sh":** fichero que permite llevar a cabo la ejecución de pruebas.

1. ARQUITECTURA DEL SISTEMA

El sistema desarrollado está basado en una arquitectura de dos capas. Por un lado el depósito de datos que es el conjunto de documentos (.xml, .bin, .txt) que intervienen en la ejecución del sistema, y por el otro, la lógica del programa que se corresponde con todos los ficheros .c y .h que realizan todo el procesamiento del sistema.

En la figura se ven representados los 2 módulos del sistema.

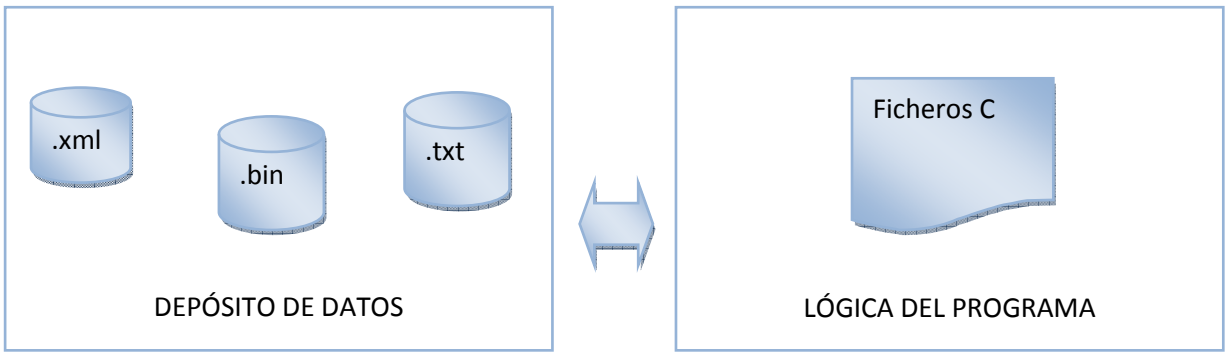

Figura 90. Arguitectura lógica del sistema.

2. DIAGRAMA DE COMPONENTES

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

A continuación se muestran los componentes del sistema y las dependencias que existen entre ellos.

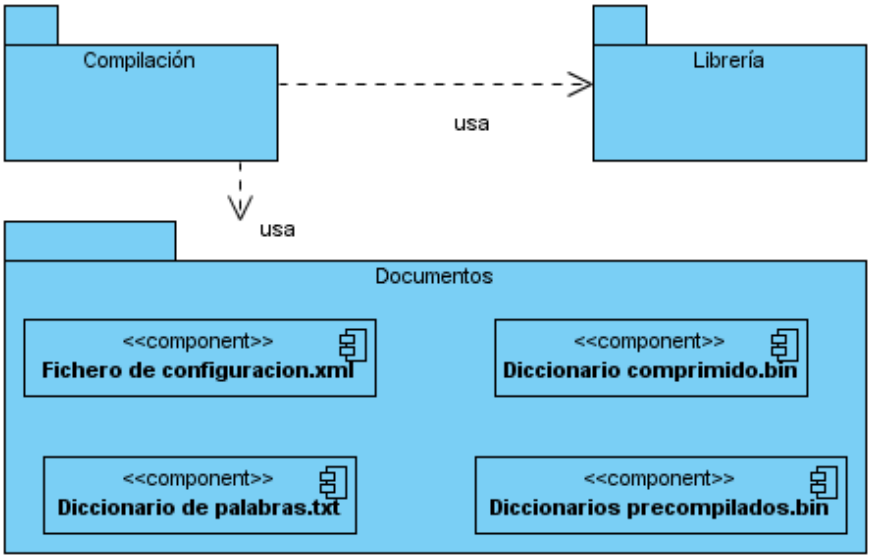

Figura 91. Diagrama de componentes

3. FICHERO XML DE CONFIGURACIÓN

El fichero empleado es un fichero XML con el siguiente .dtd asociado:

```
<!ELEMENT setup (input_automaton|info|key)*>
<!ATTLIST setup
   output_binary_file CDATA #IMPLIED
   path CDATA #IMPLIED
 >

<!ELEMENT key EMPTY>
<!ATTLIST key
   number_mappings CDATA #IMPLIED
   words_file CDATA #IMPLIED
   file CDATA #IMPLIED
 >
```

```
<!ELEMENT info EMPTY>
<!ATTLIST info
   filei CDATA #IMPLIED
   type CDATA #IMPLIED
   automaton_id CDATA #IMPLIED
   function CDATA #IMPLIED
   name CDATA #IMPLIED
 >

<!ELEMENT input_automaton EMPTY>
<!ATTLIST input_automaton
   file CDATA #IMPLIED
   automaton_id CDATA #IMPLIED
 >
```

Con los siguientes atributos:

<setup path="**ruta**" output_binary_file="**resultado.bin**">

<key file="**entrada.txt**" words_file="**palabras.txt**" forms_file="**formas.txt**"
number_mappings="**número_mappings**" />

<info name="**nombre_información**" function="**nombre_función**"
automaton_id="**número_identificador**" type="**tipo**" filei="**fichero_información.txt**" />

<input_automaton automaton_id="**número_identificador**"
file="**automata_fichero_información.bin**" />

</setup>

En donde:

- **path:** ruta donde se encuentran todos los ficheros necesarios.
- **output_binary_file:** fichero binario donde se almacena la versión compilada del diccionario.

- **Para el tag "key":**
  - **file:** fichero que contiene el diccionario original con la lista de palabras (primera columna) seguida de tantas columnas como tipos de información tenga asociado el diccionario.
  - **words_file:** fichero que contiene la lista de palabras del diccionario.

- **forms_file:** fichero donde se almacena la lista de formas de las palabras del diccionario.
- **number_mappings:** número de índices a emplear para indexar la información de las palabras (para maximizar la eficiencia del compilador se recomienda emplear 2 ó 0 en caso de no tener asociada información para las palabras).

- **Para el tag "info" (uno por cada columna de información):**
  - **name:** nombre de la columna de información.
  - **function:** función a aplicar a los datos de la columna de información, en caso de que sea necesario, es decir, si los datos a almacenar en el proceso de compilación no son los datos de las columnas de información, sino que debe hacerse algún proceso de conversión (si este campo no es necesario se emplea la cadena vacía).
  - **automaton_id:** identificador del autómata previamente compilado. Es necesario un autómata previamente compilado si los datos de la columna de información no son los que se van a almacenar en el proceso de compilación, si no que se almacenan los índices de tales datos en su autómata correspondiente para agilizar el proceso y emplear menos espacio en memoria (si este campo no es necesario se emplea la cadena vacía).
  - **type:** tipo de dato de la columna de información a almacenar en el proceso de compilación.
  - **filei:** fichero en el que se almacena cada columna de información en su versión comprimida.

- **Para el tag "input_automaton" (uno por cada autómata previamente compilado necesario):**
  - **automaton_id:** identificador del autómata previamente compilado asociado a una columna de información.
  - **file:** fichero binario con la versión comprimida de dicho autómata.

Ejemplo con los valores para un fichero de configuración de prueba:

```
<setup path="/home/nieves/libnadfa/src" output_binary_file="result.bin">
<key file="entry.txt" words_file="words.txt" forms_file="forms.txt" number_mappings="2" />
<info name="tag" function="word_to_index" automaton_id="1" type="guint32" filei="tags.txt" />
<info name="probability" function="" automaton_id="" type="gdouble" filei="probabilities.txt" />
<info name="lemma" function="word_to_index" automaton_id="2" type="guint32"
filei="lemmas.txt"/>
<input_automaton automaton_id="1" file="automaton_tags.bin" />
<input_automaton automaton_id="2" file="automaton_lemmas.bin" />
</setup>
```

- **path:** /home/nieves/libnadfa/src es la ruta donde se encuentran los ficheros.
- **output_binary_file:** la versión comprimida del diccionario se almacena en el fichero "result.bin".

- **Para el tag "key":**
  - **file:** "entry.txt" es el fichero que contiene el diccionario original.

- **words_file:** "words.txt" es el fichero que contiene la lista de palabras del diccionario.
- **forms_file:** "forms.txt" es fichero donde se almacena la lista de formas del diccionario.
- **number_mappings:** se emplean 2 índices para indexar la información de las palabras.

- **Para el tag "info" (en este caso hay 3 columnas de información):**
  - **name:** "tag", "probability" y "lemma" son los nombres de las columnas de información.
  - **function:** para las columnas "tag" y "lemma" los datos a almacenar son los que resultan de aplicar la función "word_to_index".
  - **automaton_id:** para las columnas "tag" y "lemma" se necesita emplear los autómatas con identificador "1" y "2" respectivamente, para poder obtener los datos a almacenar.
  - **type:** los tipos de datos en este caso son "guint32" y "gdouble".
  - **filei:** los ficheros donde se almacenan las columnas de información son "tags.txt", "probabilities.txt" y "lemmas.txt" respectivamente.

- **Para el tag "input_automaton" (2 autómatas en este caso):**
  - **automaton_id:** en este caso hay dos identificadores (1 y 2).
  - **file:** el autómata con identificador "1" se almacena en el fichero "automaton_tags.bin" y el autómata con identificador "2" se almacena en el fichero "automaton_lemmas.bin".

PRUEBAS

Las pruebas de la aplicación se han realizado de forma paralela a la fase de implementación, a medida que se iban completando las partes del sistema. Para realizarlas se ha empleado un léxico de 742.925 palabras obtenidas del proyecto CORGA (*Corpus de referencia do galego actual*) (16) del Centro Ramón Piñeiro para a Investigación en Humanidades (17).

Dado que, por motivos de permisos no se ha podido incluir el léxico completo, también se ha trabajado con un subconjunto aleatorio del primero constituido por 8.334 palabras.

Las pruebas han sido realizadas en relación a las dos partes del sistema:

1. Compilación diccionarios.
2. Empleo de las funciones de la librería de manejo de diccionarios.

1. Rendimiento

Las pruebas en relación al rendimiento se centran en el tiempo de compilación y reconocimiento de palabras. Las pruebas se han llevado a cabo en un equipo con procesador de 2,41 Ghz y 3,50 GB de RAM con la distribución Ubuntu 8.10.

1.1 Compilación

Las pruebas de compilación han dado resultados satisfactorios, ya que se han obtenido unos tiempos y tamaños de diccionario razonables:

Muestra	Tamaño del diccionario			Tiempos de Compilación (en segundos)	Tamaño diccionario compilado
	Palabras	Formas diferentes	bytes		
CORGA	742.925	491.796	19,3 MiB	28,24	8,1 MiB
Aleatoria	8.334	5.538	211 KiB	0,20	103,0 KiB

Figura 92. Tiempos prueba compilación.

Puede observarse que el tamaño del diccionario compilado es muy inferior al original, y cuanto mayor es el diccionario original mayor es el nivel de compresión.

1.2 Tasa de Reconocimiento

Esta prueba calcula el tiempo que tarda el sistema en reconocer una palabra, es decir, en localizarla en el autómata, mediante word_to_index(palabra). Los resultados obtenidos son satisfactorios.

Muestra	Tiempo de Reconocimiento Diccionario (en segundos)	Tiempo de Reconocimiento por palabra (en segundos)
CORGA	0,33	$6{,}7.10^{-7}$
Aleatoria	0,18	$3{,}2.10^{-5}$

Puede observarse que la tasa de reconocimiento por palabra aumenta cuanto mayor es el tamaño del diccionario.

2. Estabilidad

Para realizar pruebas con el manejo de las funciones de la librería, se ha diseñado un programa con el proceso de pruebas específico en el fichero "testing.sh":

- Proceso de compilación: primero se compila la biblioteca de diccionarios que se necesitarán posteriormente en la compilación del diccionario principal, resultados mostrados en la siguiente figura:

Muestra	Tamaño del diccionario		Tiempos de Compilación (en segundos)
	Palabras	Nº diccionarios	
CORGA	742.925	2	**4,39**
Aleatoria	**8.334**	**2**	**0,02**

Figura 93. Tiempos prueba compilación autómatas previamente compilados.

- Prueba 1: compila el diccionario principal, recupera la versión compilada en fichero binario, lo carga en memoria e imprime las palabras del autómata comprobando que la lista de palabras obtenida como resultado sea la misma que la lista de palabras de entrada.

- Prueba 2: compila el diccionario principal, recupera la versión compilada en fichero binario, lo carga en memoria y analiza las palabras de entrada una a una, compara que el índice obtenido de la función word_to_index (palabra) permite obtener la misma palabra que con la función index_to_word(índice). Seguidamente imprime la lista de palabras obtenida y compara que sea la misma que la lista de palabras de entrada.

- Prueba 3: compila el diccionario principal, recupera la versión compilada en fichero binario, lo carga en memoria y seguidamente analiza una pequeña lista de 18 palabras (seleccionadas aleatoriamente) de las que muestra toda la información asociada a ellas en el diccionario. Previamente ya se ha creado un fichero con el resultado que debería obtenerse para poder comparar si el obtenido es el resultado esperado.

Lista de palabras:

```
apendicular
desestimo
```

```
!
B<subíndice>12</subíndice>
H<subíndice>2</subíndice>O<subíndice>2</subíndice>
asalariaba
a
acarreirabades
A. B. D.
acarreirabamos
foguease
c
zurzan
b
A-V
acarreiraba
úvulas
ineeeexistente
```

Figura 94. Listado de prueba 3.

El resultado obtenido es el esperado:

```
word: apendicular
information: lemma
apendicular
apendicular
apendicular
first_information: 65
last_information: 67

word: desestimo
information: lemma
desestimar
first_information: 201
last_information: 201

word: !
information: lemma
!
first_information: 1
last_information: 1

word: B<subíndice>12</subíndice>
information: lemma
B<subíndice>12</subíndice>
first_information: 35
last_information: 35

word: H<subíndice>2</subíndice>O<subíndice>2</subíndice>
information: lemma
H<subíndice>2</subíndice>O<subíndice>2</subíndice>
first_information: 43
last_information: 43

word: asalariaba
information: lemma
asalariar
asalariar
asalariar
```

```
first_information: 106
last_information: 108

word: a
information: lemma
o
a
o
a
first_information: 44
last_information: 47

word: acarreirabades
information: lemma
acarreirar
first_information: 51
last_information: 51

word: A. B. D.
information: lemma
A. B. D.
A. B. D.
A. B. D.
A. B. D.
A. B. D.
first_information: 18
last_information: 22

word: acarreirabamos
information: lemma
acarreirar
first_information: 52
last_information: 52

word: foguease
information: lemma
foguear
foguear
foguear
first_information: 212
last_information: 214

word: c
information: lemma
c
c
c
first_information: 139
last_information: 141

word: zurzan
information: lemma
zurcir
first_information: 8328
last_information: 8328

word: b
information: lemma
b
b
b
```

```
first_information: 133
last_information: 135

word: A-V
information: lemma
A-V
A-V
A-V
A-V
A-V
first_information: 10
last_information: 14

word: acarreiraba
information: lemma
acarreirar
acarreirar
acarreirar
first_information: 48
last_information: 50

word: úvulas
information: lemma
úvula
first_information: 8334
last_information: 8334

word: ineeeexistente
information: lemma
(getInfo) THE WORD ineeeexistente ISN'T IN THE DICTIONARY.
first_information: 0
last_information: 0
```

Figura 95. Resultado prueba 3.

- Prueba 4: compila el diccionario principal, recupera la versión comprimida en fichero binario, lo carga en memoria y seguidamente analiza una pequeña lista de 18 palabras (la misma que en la prueba anterior) de las que muestra una información determinada (el lema[8] en este caso), además muestra los índices de la primera y la última información de cada palabra. Previamente ya se ha creado un fichero con el resultado que debería obtenerse para poder comparar si el obtenido es el resultado esperado.

El resultado obtenido al igual que en el resto de los casos es el esperado:

```
word: apendicular
tag: A0as         lemma: apendicular
tag: A0fs         lemma: apendicular
tag: A0ms         lemma: apendicular

word: desestimo
tag: Vpi10s       lemma: desestimar

word: !
tag: Q!           lemma: !
```

[8] Abstracción, a partir del haz de rasgos flexivos de una palabra, que representa a esta como forma canónica.

```
word: B<subíndice>12</subíndice>
tag: Zs00         lemma: B<subíndice>12</subíndice>

word: H<subíndice>2</subíndice>O<subíndice>2</subíndice>
tag: Zs00         lemma: H<subíndice>2</subíndice>O<subíndice>2</subí-
ndice>

word: asalariaba
tag: Vii10s       lemma: asalariar
tag: Vii30s       lemma: asalariar
tag: Viia0s       lemma: asalariar

word: a
tag: Ddfs         lemma: o
tag: P            lemma: a
tag: Raa3fs       lemma: o
tag: Scms         lemma: a

word: acarreirabades
tag: Vii20p       lemma: acarreirar

word: A. B. D.
tag: Zg00         lemma: A. B. D.
tag: Zgfp         lemma: A. B. D.
tag: Zgfs         lemma: A. B. D.
tag: Zgmp         lemma: A. B. D.
tag: Zgms         lemma: A. B. D.

word: acarreirabamos
tag: Vii10p       lemma: acarreirar

word: foguease
tag: Ves10s       lemma: foguear
tag: Ves30s       lemma: foguear
tag: Vesa0s       lemma: foguear

word: c
tag: Scma         lemma: c
tag: Scmp         lemma: c
tag: Scms         lemma: c

word: zurzan
tag: Vps30p       lemma: zurcir

word: b
tag: Scma         lemma: b
tag: Scmp         lemma: b
tag: Scms         lemma: b

word: A-V
tag: Zg00         lemma: A-V
tag: Zgfp         lemma: A-V
tag: Zgfs         lemma: A-V
tag: Zgmp         lemma: A-V
tag: Zgms         lemma: A-V

word: acarreiraba
tag: Vii10s       lemma: acarreirar
tag: Vii30s       lemma: acarreirar
tag: Viia0s       lemma: acarreirar
```

```
word: úvulas
tag: Scfp          lemma: úvula

word: ineeeexistente
      (getInfos) THE WORD ineeeexistente ISN'T IN THE DICTIONARY.
```

Figura 96. Resultado prueba 4.

A continuación se muestran las capturas de pantalla de las palabras (de prueba) en el diccionario, mediante las que se puede comprobar que los resultados obtenidos son satisfactorios, teniendo en cuenta que cada línea sigue el formato:

nombre fichero **:** índice palabra **:** ***palabra*** ***etiqueta*** ***lema***

Palabra	Captura
apendicular	./entry.txt: 65: apendicular A0as apendicular ./entry.txt: 66: apendicular A0fs apendicular ./entry.txt: 67: apendicular A0ms apendicular
desestimo	./entry.txt: 201: desestimo Vpi10s desestimar
!	./entry.txt: 1: ! Q! !
B<subíndice>12</subíndice>	./entry.txt: 35: B<subíndice>12</subíndice> Zs00 B<subíndice>12</subíndice>
H<subíndice>2</subíndice>0 <subíndice>2</subíndice>	./entry.txt: 43: H<subíndice>2</subíndice>0<subíndice>2</subíndice> Zs00 H<subíndice>2</subíndice>0<subíndice>2</subíndice>
asalariaba	./entry.txt: 106: asalariaba Vii10s asalariar ./entry.txt: 107: asalariaba Vii30s asalariar ./entry.txt: 108: asalariaba Viia0s asalariar
a	./entry.txt: 44: a Ddfs o ./entry.txt: 45: a P a ./entry.txt: 46: a Raa3fs o ./entry.txt: 47: a Scms a
acarreirabades	./entry.txt: 51: acarreirabades Vii20p acarreirar
A. B. D.	./entry.txt: 18: A. B. D. Zg00 A. B. D. ./entry.txt: 19: A. B. D. Zgfp A. B. D. ./entry.txt: 20: A. B. D. Zgfs A. B. D. ./entry.txt: 21: A. B. D. Zgmp A. B. D. ./entry.txt: 22: A. B. D. Zgms A. B. D.
acarreirabamos	./entry.txt: 52: acarreirabamos Vii10p acarreirar

foguease	./entry.txt: 212: foguease		Ves10s	foguear
	./entry.txt: 213: foguease		Ves30s	foguear
	./entry.txt: 214: foguease		Vesa0s	foguear
c	./entry.txt: 139: c	Scma	c	
	./entry.txt: 140: c	Scmp	c	
	./entry.txt: 141: c	Scms	c	
zurzan	./entry.txt: 8328: zurzan		Vps30p	zurcir
b	./entry.txt: 133: b	Scma	b	
	./entry.txt: 134: b	Scmp	b	
	./entry.txt: 135: b	Scms	b	
A-V	./entry.txt: 10: A-V	Zg00	A-V	
	./entry.txt: 11: A-V	Zgfp	A-V	
	./entry.txt: 12: A-V	Zgfs	A-V	
	./entry.txt: 13: A-V	Zgmp	A-V	
	./entry.txt: 14: A-V	Zgms	A-V	
acarreiraba	./entry.txt: 48: acarreiraba		Vii10s	acarreirar
úvulas	./entry.txt: 8334: úvulas		Scfp	úvula

MANUAL USUARIO

INTRODUCCIÓN

Acerca del uso de este proyecto.

En este apartado se da información acerca de los requisitos, instalación y puesta en funcionamiento del compilador y la librería desarrollados, además de la guía sobre el uso de los mismos.

En primer lugar, se describen los requisitos de hardware y software necesarios para un correcto funcionamiento del sistema desarrollado.

Seguidamente, se detallan los pasos a seguir para la instalación y configuración del software necesario para el correcto funcionamiento del sistema.

El manual está dirigido a todas aquellas personas interesadas en el uso de esta librería, con el objetivo de guiarles en los pasos que son necesarios para su uso. El sistema desarrollado está enfocado para un usuario con perfil de programador, ya que será el encargado de administrar y mantener la librería, además de realizar las modificaciones que crea convenientes. Es necesario que el programador tenga conocimientos de C para poder entender el funcionamiento del sistema y poder adaptarlo a sus necesidades.

REQUISITOS

Para poder poner en marcha el sistema.

En este apartado se describen los requisitos mínimos en cuanto a hardware y software para el correcto funcionamiento de la librería.

1. REQUISITOS HARDWARE

Los requisitos mínimos y recomendados en cuanto al hardware se indican a continuación:

- Procesador Intel Pentium III a 1000 Mhz o superior
- Memoria RAM a 216 MB o superior
- Espacio en disco duro de 1GB
- Unidad de CD-ROM
- Monitor
- Teclado

2. REQUISITOS SOFTWARE

Los requisitos mínimos en cuanto al software son:

- Sistema operativo GNU/Linux para poder ejecutar aquellas herramientas desarrolladas para el caso concreto y garantizar un entorno robusto, estable y rápido. Se recomienda una distribución basada en Ubuntu.
- Compilador de C.
- Librería glib 2.12 que asegura la portabilidad entre sistemas operativos.
- Librería libxml2 para el acceso a los archivos XML de configuración.
- Automake/autoconf tools.

COMPILACIÓN

Cómo compilar los ficheros que componen la implementación del sistema.

Para poder compilar los ficheros del sistema el programador debe situarse en el directorio raíz desde consola y ejecutar las siguientes instrucciones:

aclocal

autoheader

autoconf

automake

./configure - -prefix = directorio/libnadfainstall

make

make install

make check

EJECUCIÓN Y FUNCIONAMIENTO DE LA LIBRERÍA

Cómo interactuar con el sistema.

A continuación se detallan los pasos a seguir para poder usar correctamente el sistema implementado.

- Para llevar a cabo la compilación de un diccionario el usuario debe introducir por consola:

  nadfac <fichero de configuración xml>

- Para emplear las funciones de la librería:

  - Si el usuario desea obtener el índice de una palabra en el diccionario → debe hacer una llamada desde su código al método **nadfa_bin_word_to_index(NADFA_BIN my_lexicon, NADFA_ALPHABET_TYPE* word)** pasándole como parámetros la estructura NADFA_BIN que contiene en memoria el diccionario compilado recuperado de fichero binario y la palabra de la que se desea obtener el índice en el autómata.

  - Si el usuario desea obtener la palabra correspondiente a un índice en el diccionario → debe hacer una llamada desde su código al método **nadfa_bin_index_to_word(NADFA_BIN my_lexicon, NADFA_NUM_WORDS_TYPE index, NADFA_ALPHABET_TYPE *word)** pasándole como parámetros la estructura NADFA_BIN que contiene en memoria el diccionario compilado recuperado de fichero binario, el índice que se desea revisar y una cadena para almacenar la palabra resultante.

  - Si el usuario desea obtener toda la información almacenada sobre una palabra → debe hacer una llamada desde su código al método **getInfos(NADFA_BIN my_lexicon, NADFA_ALPHABET_TYPE* word, NADFA_BIN* lexicon)** pasándole como parámetros la estructura NADFA_BIN que contiene en memoria el diccionario compilado recuperado de fichero binario, la palabra de la que desea obtener la información y la variable que almacena el conjunto de autómatas previamente compilados.

    La función devuelve una cadena con la información separada por tabuladores para las columnas y por saltos de línea para las filas.

  - Si el usuario desea obtener una información concreta sobre una palabra del diccionario → debe hacer una llamada desde su código al método **getInfo(NADFA_BIN my_lexicon, NADFA_ALPHABET_TYPE* word, NADFA_BIN* lexicon, NADFA_ALPHABET_TYPE* info)** pasándole como parámetros la estructura NADFA_BIN que contiene en memoria el diccionario compilado recuperado de fichero binario, la palabra de la que desea obtener la información, la variable que

almacena el conjunto de autómatas previamente compilados y el nombre de la información que se desea obtener.

La función devuelve una cadena con la información separada por saltos de línea.

- Si desea obtener el índice de la primera información de una palabra → debe hacer una llamada desde su código al método **nadfa_bin_get_first_info_index (NADFA_BIN my_lexicon, NADFA_ALPHABET_TYPE* word)** pasándole como parámetros la estructura NADFA_BIN que contiene en memoria el diccionario compilado recuperado de fichero binario y la palabra de la que desea obtener el índice.

- Si desea obtener el índice de la última aparición de una palabra → debe hacer una llamada desde su código al método **nadfa_bin_get_last_info_index(NADFA_BIN my_lexicon, NADFA_ALPHABET_TYPE* word)** pasándole como parámetros la estructura NADFA_BIN que contiene en memoria el diccionario compilado recuperado de fichero binario y la palabra de la que desea obtener el índice.

- Si el programador desea cargar en memoria un diccionario compilado → debe hacer una llamada desde su código al método **nadfa_bin_load_nadfa_bin(NADFA_BIN *lexicon, NADFA_FILE* file_name)** pasándole como parámetros la variable que almacenará los datos del diccionario en memoria, y el nombre del fichero que contiene el diccionario que se quiere cargar .

- Si el programador desea liberar la memoria ocupada por un diccionario → debe hacer una llamada al método **nadfa_bin_destroy_nadfa_bin(NADFA_BIN* lexicon)** que libera la memoria ocupada por el diccionario pasado como parámetro.

1. Fichero de configuración XML

El fichero empleado es un fichero XML con el siguiente .dtd asociado:

```
<!ELEMENT setup (input_automaton|info|key)*>
<!ATTLIST setup
   output_binary_file CDATA #IMPLIED
   path CDATA #IMPLIED
 >

<!ELEMENT key EMPTY>
<!ATTLIST key
```

```
    number_mappings CDATA #IMPLIED
    words_file CDATA #IMPLIED
    file CDATA #IMPLIED
  >

<!ELEMENT info EMPTY>
<!ATTLIST info
    filei CDATA #IMPLIED
    type CDATA #IMPLIED
    automaton_id CDATA #IMPLIED
    function CDATA #IMPLIED
    name CDATA #IMPLIED
  >

<!ELEMENT input_automaton EMPTY>
<!ATTLIST input_automaton
    file CDATA #IMPLIED
    automaton_id CDATA #IMPLIED
  >
```

Con los siguientes atributos:

<setup path="**ruta**" output_binary_file="**resultado.bin**">

<key file="**entrada.txt**" words_file="**palabras.txt**" forms_file=**"formas.txt"**
number_mappings="**número_mappings**" />

<info name="**nombre_información**" function="**nombre_función**"
automaton_id="**número_identificador**" type="**tipo**" filei="**fichero_información.txt**" />

<input_automaton automaton_id="**número_identificador**"
file="**automata_fichero_información.bin**" />

</setup>

En donde:

- **path:** ruta donde se encuentran todos los ficheros necesarios.
- **output_binary_file:** fichero binario donde se almacena la versión compilada del diccionario.

- **Para el tag "key":**
  - **file:** fichero que contiene el diccionario original con la lista de palabras (primera columna) seguida de tantas columnas como tipos de información tenga asociado el diccionario.
  - **words_file:** fichero que contiene la lista de palabras del diccionario.
  - **forms_file:** fichero donde se almacena la lista de formas de las palabras del diccionario.
  - **number_mappings:** número de índices a emplear para indexar la información de las palabras (para maximizar la eficiencia del compilador se recomienda emplear 2 ó 0 en caso de no tener asociada información para las palabras).

- **Para el tag "info" (uno por cada columna de información):**
  - **name:** nombre de la columna de información.
  - **function:** función a aplicar a los datos de la columna de información, en caso de que sea necesario, es decir, si los datos a almacenar en el proceso de compilación no son los datos de las columnas de información, sino que debe hacerse algún proceso de conversión (si este campo no es necesario se emplea la cadena vacía).
  - **automaton_id:** identificador del autómata previamente compilado. Es necesario un autómata previamente compilado si los datos de la columna de información no son los que se van a almacenar en el proceso de compilación, si no que se almacenan los índices de tales datos en su autómata correspondiente para agilizar el proceso y emplear menos espacio en memoria (si este campo no es necesario se emplea la cadena vacía).
  - **type:** tipo de dato de la columna de información a almacenar en el proceso de compilación.
  - **filei:** fichero en el que se almacena cada columna de información en su versión comprimida.

- **Para el tag "input_automaton" (uno por cada autómata previamente compilado necesario):**
  - **automaton_id:** identificador del autómata previamente compilado asociado a una columna de información.
  - **file:** fichero binario con la versión comprimida de dicho autómata.

Ejemplo con los valores para un fichero de configuración de prueba:

```
<setup path="/home/nieves/libnadfa/src" output_binary_file="result.bin">
<key file="entry.txt" words_file="words.txt" forms_file="forms.txt" number_mappings="2" />
<info name="tag" function="word_to_index" automaton_id="1" type="guint32" filei="tags.txt" />
<info name="probability" function="" automaton_id="" type="gdouble" filei="probabilities.txt" />
<info name="lemma" function="word_to_index" automaton_id="2" type="guint32"
filei="lemmas.txt"/>
<input_automaton automaton_id="1" file="automaton_tags.bin" />
<input_automaton automaton_id="2" file="automaton_lemmas.bin" />
</setup>
```

- **path:** /home/nieves/libnadfa/src es la ruta donde se encuentran los ficheros.

- **output_binary_file:** la versión comprimida del diccionario se almacena en el fichero "result.bin".

- **Para el tag "key":**
  - **file:** "entry.txt" es el fichero que contiene el diccionario original.
  - **words_file:** **"**words.txt" es el fichero que contiene la lista de palabras del diccionario.
  - **forms_file:** "forms.txt" es fichero donde se almacena la lista de formas del diccionario.
  - **number_mappings:** se emplean 2 índices para indexar la información de las palabras.

- **Para el tag "info" (en este caso hay 3 columnas de información):**
  - **name:** **"**tag", "probability" y "lemma" son los nombres de las columnas de información.
  - **function:** para las columnas "tag" y "lemma" los datos a almacenar son los que resultan de aplicar la función "word_to_index".
  - **automaton_id:** para las columnas "tag" y "lemma" se necesita emplear los autómatas con identificador "1" y "2" respectivamente, para poder obtener los datos a almacenar.
  - **type:** los tipos de datos en este caso son "guint32" y "gdouble".
  - **filei:** los ficheros donde se almacenan las columnas de información son "tags.txt", "probabilities.txt" y "lemmas.txt" respectivamente.

- **Para el tag "input_automaton" (2 autómatas en este caso):**
  - **automaton_id:** en este caso hay dos identificadores (1 y 2).
  - **file:** el autómata con identificador "1" se almacena en el fichero "automaton_tags.bin" y el autómata con identificador "2" se almacena en el fichero "automaton_lemmas.bin".

BIBLIOGRAFÍA

1. **Marín, Roque.** Teoría de autómatas y lenguajes formales. [En línea] Julio de 2003. [Citado el: 30 de 10 de 2008.] perseo.dif.um.es/~roque/talf/Material/Tema3-printout.pdf.

2. **Daciuk, J., y otros.** *Incremental Construction of Minimal Acyclic Finite-State Automata Computational Linguistics.* 2000. págs. 3-16. Vol. 26(1).

3. **Graña Gil, J., Barcala Rodríguez, F.M. y Alonso Pardo, M.A.** *Compilation Methods of Minimal Acyclic Finite-State Automata for Large Dictionaries.* Pretoria, South Africa : s.n., 2001. págs. 135-148. In Proc. of CIAA-2001.

4. XML, World Wide Web Consortium. [En línea] [Citado el: 1 de 10 de 2008.] http://www.w3c.org.

5. **Schildt, Herbert.** *Turbo C/C++ Manual de referencia .* s.l. : Osborne McGraw-Hill , 1992.

6. Biblioteca de Documentación de GNOME. [En línea] [Citado el: 10 de 11 de 2008.] http://library.gnome.org/devel/glib/.

7. libxml2. [En línea] [Citado el: 1 de 11 de 2008.] http://xmlsoft.org/.

8. **Gragy Booch, James Rumbaugh, Ivar Jacobson.** *El lenguaje Unificado de Modelado.* s.l. : Addison Wesley.

9. **Miguel Arregui, Depto. De lenguajes y sistemas informáticos, Grupo Iris (integración y reingeniería de sistemas), Universitat Jaume I.** Tutorial de UML. [En línea] [Citado el: 27 de 10 de 2008.] www.conganat.org/Seis/inforsalud04/2004_Inforsalud_TutorialUML-UP.doc.

10. Wikipedia, la enciclopedia libre. [En línea] 14 de 10 de 2008. http://es.wikipedia.org/wiki/Aut%C3%B3mata_finito.

11. **Aho, A.V., Sethi, R. y Ullman, J.D.** *Compilers: principles, techniques and tools.* s.l. : Addison-Wesley.

12. Wikipedia, la enciclopedia libre . [En línea] [Citado el: 23 de 9 de 2008.] http://es.wikipedia.org/wiki/Biblioteca_C#Biblioteca_C.

13. Wikipedia, la enciclopedia libre . [En línea] [Citado el: 31 de 10 de 2008.] http://es.wikipedia.org/wiki/GLib.

14. Wikipedia, la enciclopedia libre . [En línea] http://en.wikipedia.org/wiki/Libxml2.

15. Wikipedia, la enciclopedia libre . [En línea] http://es.wikipedia.org/wiki/XML.

16. CORGA: Corpus de Referencia do Galego Actual. [En línea] http://corpus.cirp.es/corgaxml.

17. Centro Ramón Piñeiro para a Investigación en Humanidades. [En línea] http://www.cirp.es.

TABLA DE FIGURAS

FIGURA 1. NIVELES DEL LENGUAJE. ---- 14
FIGURA 2. ARQUITECTURA DEL PROCESO DE COMPILACIÓN. ---- 18
FIGURA 3. ESTRUCTURA INTERNA DEL SISTEMA. ---- 19
FIGURA 4. ESTIMACIÓN TEMPORAL INICIAL.---- 21
FIGURA 5. DURACIÓN ESTIMADA.---- 21
FIGURA 6. DIAGRAMA DE GANTT INICIAL.---- 22
FIGURA 7. DURACIÓN REAL.---- 22
FIGURA 8. DIAGRAMA DE GANTT FINAL ---- 23
FIGURA 9. ESTRUCTURAS UTILIZADAS POR EL COMPILADOR. ---- 31
FIGURA 10. DIAGRAMA DE CASOS DE USO DEL SISTEMA ---- 36
FIGURA 11. DIAGRAMA DE SECUENCIA DEL CASO DE USO COMPILACIÓN DICCIONARIO (ANÁLISIS). ---- 37
FIGURA 12. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO COMPILACIÓN DICCIONARIO (ANÁLISIS). ---- 38
FIGURA 13. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO COMPILACIÓN DICCIONARIO (ANÁLISIS).---- 39
FIGURA 14. DIAGRAMA DE SECUENCIA DEL CASO DE USO CARGAR DICCIONARIO EN MEMORIA (ANÁLISIS). ---- 41
FIGURA 15. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO CARGAR DICCIONARIO EN MEMORIA (ANÁLISIS). ---- 41
FIGURA 16. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO CARGAR DICCIONARIO COMPILADO EN MEMORIA (ANÁLISIS). --- 42
FIGURA 17. DIAGRAMA DE SECUENCIA DEL CASO DE USO CONVERSIÓN PALABRA A ÍNDICE (ANÁLISIS). ---- 43
FIGURA 18. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO CONVERSIÓN PALABRA A ÍNDICE (ANÁLISIS). ---- 43
FIGURA 19. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO CONVERSIÓN PALABRA A ÍNDICE (ANÁLISIS) ---- 44
FIGURA 20. DIAGRAMA DE SECUENCIA DEL CASO DE USO CONVERSIÓN ÍNDICE A PALABRA (ANÁLISIS). ---- 45
FIGURA 21. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO CONVERSIÓN ÍNDICE A PALABRA (ANÁLISIS). ---- 45
FIGURA 22. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO CONVERSIÓN ÍNDICE A PALABRA (ANÁLISIS). ---- 45
FIGURA 23. DIAGRAMA DE SECUENCIA DEL CASO DE USO ACCEDER INFORMACIÓN COMPLETA DE PALABRA (ANÁLISIS).----- 47
FIGURA 24. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ACCEDER INFORMACIÓN COMPLETA DE PALABRA (ANÁLISIS). 47
FIGURA 25. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ACCEDER INFORMACIÓN COMPLETA DE PALABRA (ANÁLISIS). -- 48
FIGURA 26. DIAGRAMA DE SECUENCIA DEL CASO DE USO ACCEDER INFORMACIÓN CONCRETA DE PALABRA (ANÁLISIS).----- 49
FIGURA 27. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ACCEDER INFORMACIÓN CONCRETA DE PALABRA (ANÁLISIS). 49
FIGURA 28. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ACCEDER INFORMACIÓN CONCRETA DE PALABRA (ANÁLISIS).--- 50
FIGURA 29. DIAGRAMA DE SECUENCIA DEL CASO DE USO ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA (ANÁLISIS). ---- 51
FIGURA 30. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA (ANÁLISIS).
---- 51
FIGURA 31. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA (ANÁLISIS).-- 52
FIGURA 32. DIAGRAMA DE SECUENCIA DEL CASO DE USO ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA (ANÁLISIS). ----- 53
FIGURA 33. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA (ANÁLISIS). 53
FIGURA 34. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA (ANÁLISIS). --- 54
FIGURA 35. DIAGRAMA DE SECUENCIA DEL CASO DE USO LIBERAR RECURSOS (ANÁLISIS). ---- 55
FIGURA 36. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO LIBERAR RECURSOS (ANÁLISIS). ---- 55
FIGURA 37. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO LIBERAR RECURSOS (ANÁLISIS). ---- 55
FIGURA 38. DIAGRAMA DE CLASES DEL SISTEMA (ANÁLISIS).---- 56
FIGURA 39. DIAGRAMA DE CLASES DEL SISTEMA (DISEÑO). ---- 57
FIGURA 40. DIAGRAMA DE CLASES PARCIAL DE LA CLASE COMPILADOR.---- 58
FIGURA 41. DIAGRAMA DE CLASES PARCIAL DE LA CLASE NADFA.---- 60
FIGURA 42. DIAGRAMA DE CLASES PARCIAL DE LA CLASE NADFA_AUTOMATA ---- 62
FIGURA 43. AUTÓMATA INICIAL QUE RECONOCE LA CADENA "ABCD" ---- 65
FIGURA 44. INSERCIÓN DE LA PALABRA "ABCDECD" SIN CONTROL DE CICLOS---- 65

FIGURA 45. INSERCIÓN DE LA PALABRA "ABCDECD" CON CONTROL DE CICLOS.---------- 66
FIGURA 46. AUTÓMATA INICIAL QUE RECONOCE LA CADENA "ABCD" Y "XSCD". ---------- 66
FIGURA 47. INSERCIÓN DE LA PALABRA "ABCSN" SIN CONTROL DE ESTADOS CONFLUENCIA. NO SOLAMENTE SE RECONOCE "ABCSN", SINO TAMBIÉN "XSCSN".---------- 66
FIGURA 48. INSERCIÓN DE LA PALABRA "ABCSN" CON CONTROL DE ESTADOS CONFLUENCIA. RECONOCE "ABCSN", "ABCD" Y "XSCD".---------- 67
FIGURA 49. REORGANIZACIÓN DE ÍNDICES EN INSERCIÓN DE PALABRA.---------- 71
FIGURA 50. DISTRIBUCIÓN DE BYTES EN EL AUTÓMATA ---------- 73
FIGURA 51. AUTÓMATA INICIAL QUE RECONOCE LA CADENA "ABB" Y "BA". ---------- 74
FIGURA 52. ESTRUCTURA COMPRIMIDA DEL AUTÓMATA.---------- 75
FIGURA 53. DIAGRAMA DE CLASES PARCIAL DE LA CLASE NADFA_MAPPINGS ---------- 76
FIGURA 54. TABLAS *MAPPING* INSERCIÓN DE UNA PALABRA. ---------- 77
FIGURA 55. TABLAS *MAPPING* PALABRA REPETIDA.---------- 77
FIGURA 56. ARRAY CON TABLAS *MAPPING* UNIFICADAS Y ORDENADAS.---------- 78
FIGURA 57. DIAGRAMA DE CLASES DE LA CLASE NADFA_INFOS---------- 79
FIGURA 58. ESTRUCTURA NADFA ---------- 81
FIGURA 59. ACCESO AUTÓMATA PRE-COMPILADO. ---------- 82
FIGURA 60. ALMACENAMIENTO ORDENADO DE LA INFORMACIÓN DEL DICCIONARIO EN FICHERO BINARIO---------- 83
FIGURA 61. DIAGRAMA DE CLASES PARCIAL DE LA CLASE NADFA_BIN ---------- 84
FIGURA 62. DIAGRAMA DE CLASES PARCIAL DE LA CLASE E/S---------- 91
FIGURA 63. DIAGRAMA DE SECUENCIA DEL CASO DE USO COMPILACIÓN DICCIONARIO (DISEÑO). ---------- 92
FIGURA 64. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO COMPILACIÓN DICCIONARIO (DISEÑO).---------- 93
FIGURA 65. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO COMPILACIÓN DICCIONARIO (DISEÑO)---------- 94
FIGURA 66. DIAGRAMA DE SECUENCIA DEL CASO DE USO CARGAR DICCIONARIO COMPILADO EN MEMORIA (DISEÑO).------- 95
FIGURA 67. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO CARGAR DICCIONARIO COMPILADO EN MEMORIA (DISEÑO). - 95
FIGURA 68. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO CARGAR DICCIONARIO COMPILADO EN MEMORIA (DISEÑO). ---- 96
FIGURA 69. DIAGRAMA DE SECUENCIA DEL CASO DE USO CONVERSIÓN PALABRA A ÍNDICE (DISEÑO).---------- 96
FIGURA 70. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO CONVERSIÓN PALABRA A ÍNDICE (DISEÑO). ---------- 97
FIGURA 71. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO CONVERSIÓN PALABRA A ÍNDICE (DISEÑO) ---------- 97
FIGURA 72. DIAGRAMA DE SECUENCIA DEL CASO DE USO CONVERSIÓN ÍNDICE A PALABRA (DISEÑO).---------- 98
FIGURA 73. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO CONVERSIÓN ÍNDICE A PALABRA (DISEÑO). ---------- 98
FIGURA 74. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO CONVERSIÓN ÍNDICE A PALABRA (DISEÑO) ---------- 99
FIGURA 75. DIAGRAMA DE SECUENCIA DEL CASO DE USO ACCEDER INFORMACIÓN COMPLETA DE PALABRA (DISEÑO). ------ 99
FIGURA 76. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ACCEDER INFORMACIÓN COMPLETA DE PALABRA (DISEÑO). 100
FIGURA 77. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ACCEDER INFORMACIÓN COMPLETA DE PALABRA (DISEÑO) --- 100
FIGURA 78. DIAGRAMA DE SECUENCIA DEL CASO DE USO ACCEDER INFORMACIÓN CONCRETA DE PALABRA (DISEÑO). ---- 101
FIGURA 79. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ACCEDER INFORMACIÓN CONCRETA DE PALABRA (DISEÑO). 101
FIGURA 80. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ACCEDER INFORMACIÓN CONCRETA DE PALABRA (DISEÑO)--- 102
FIGURA 81. DIAGRAMA DE SECUENCIA DEL CASO DE USO ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA (DISEÑO). --- 102
FIGURA 82. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ÍNDICE PRIMERA INFORMACIÓN DE UNA PALABRA (DISEÑO). ---------- 103
FIGURA 83. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ÍNDICE PRIMERA INFORMACIÓN POR PALABRA (DISEÑO) ------ 103
FIGURA 84. DIAGRAMA DE SECUENCIA DEL CASO DE USO ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA (DISEÑO).----- 104
FIGURA 85. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA (DISEÑO). 104
FIGURA 86. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO ÍNDICE ÚLTIMA INFORMACIÓN DE UNA PALABRA (DISEÑO)---- 105
FIGURA 87. DIAGRAMA DE SECUENCIA DEL CASO DE USO LIBERAR RECURSOS (DISEÑO).---------- 105
FIGURA 88. DIAGRAMA DE COLABORACIÓN DEL CASO DE USO LIBERAR RECURSOS (DISEÑO). ---------- 106
FIGURA 89. DIAGRAMA DE ACTIVIDADES DEL CASO DE USO LIBERAR RECURSOS (DISEÑO). ---------- 106
FIGURA 90. ARQUITECTURA LÓGICA DEL SISTEMA. ---------- 108
FIGURA 91. DIAGRAMA DE COMPONENTES ---------- 109

FIGURA 92. TIEMPOS PRUEBA COMPILACIÓN. ---------- 113
FIGURA 92. TIEMPOS PRUEBA COMPILACIÓN AUTÓMATAS PREVIAMENTE COMPILADOS.---------- 114
FIGURA 93. LISTADO DE PRUEBA 3. ---------- 115
FIGURA 94. RESULTADO PRUEBA 3. ---------- 117
FIGURA 95. RESULTADO PRUEBA 4. ---------- 119