UNIVERSIDAD
DE **V**IGO

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

Manual Técnico y de Usuario del Proyecto Fin de Carrera que presenta

Doña Milagros Fernández Gavilanes

para la obtención del Título de **Enxeñeiro en Informática**

Diseño e Implementación de una Arquitectura Genérica para Desarrollo de Sistemas de Gestión de Documentos Textuales

Septiembre, 2005

Proyecto Fin de Carrera Nº: ENI-118

Director: Francisco José Ribadas Pena
Área de conocimiento: Ciencias da Computación e Intelixencia Artificial
Departamento: Informática

Contenido

Índice de Figuras

MANUAL TÉCNICO

1. Introducción

1.1 Identificación del Proyecto

Código del PFC: **ENI 118**

Título del PFC: **Diseño e implementación de una arquitectura genérica para el desarrollo de sistemas de gestión de documentos textuales.**

Alumna: **Milagros Fernández Gavilanes**

Director del proyecto: **Francisco José Ribadas Pena**

Departamento: **Informática**

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

Titulación: **Ingeniería Informática**

1.2 Organización de la Documentación del Proyecto

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

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

1.2.1 Manual Técnico

Este manual Técnico incluye los siguientes capítulos:

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

2. *Desarrollo del proyecto*: En este capítulo se muestra los datos estimados y reales sobre la planificación temporal y costes finales.
3. *Análisis y Diseño de la arquitectura*: Contiene la especificación de los requerimientos de la arquitectura desarrollada y la descripción de todos los casos de uso así como los diagramas de secuencia y de colaboración obtenidos a través de los escenarios. También se realiza un posterior refinamiento de los diagramas, obteniendo un mayor nivel de especificación del sistema.
4. *Análisis y Diseño de la aplicación Arquitex*: Contiene la especificación de los requerimientos de la aplicación de ejemplo de la arquitectura desarrollada y la descripción de todos los casos de uso así como los diagramas de secuencia y de colaboración obtenidos a través de los escenarios. También se incluyen los diagramas de clases parciales para cara cada uno de los escenarios.
5. *Implementación y pruebas*: Este capítulo contiene la descripción de la solución adoptada para cada una de las partes del proyecto, junto con ejemplos de código fuente y las pruebas realizadas para verificar la integración, la validez y eficiencia del sistema.
6. *Problemas, conclusiones y posibles ampliaciones*: Contiene una relación de los problemas encontrados durante el desarrollo del proyecto, las ventajas que aporta el sistema implementado y las posibles ampliaciones que se podrían acometer en un futuro.
7. *Apéndices técnicos*: En él se documentan las tecnologías o aspectos técnicos más relevantes que se usaron durante el desarrollo del proyecto.

1.2.2 Manual de Usuario

El manual de usuario se compone de los manuales de uso de la **arquitectura** y del uso de la **aplicación**.

En el **manual de la arquitectura** indica como se deberá llevar a cabo, por parte de los desarrolladores que hagan uso de la arquitectura propuesta, la implementación de las distintas clases que pueden hacer uso de la misma y sus componentes: clases decoradoras, clases de servicios, clases cargadoras y clases de visualización.

El manual de **usuario de la aplicación** ofrece una descripción del funcionamiento y del uso de la aplicación desarrollada para explicar de un modo sencillo el funcionamiento de la arquitectura genérica implementada. Incluye toda la información necesaria para llevar a cabo la instalación del sistema y las instrucciones para su correcta utilización. Por lo tanto dicho manual irá dirigido tanto a personal técnico como a usuarios finales del mismo. Los principales puntos que se van a tratar en este manual son:

1. *Requisitos Hardware y software*: descripción de los requisitos mínimos necesarios para el correcto funcionamiento de la aplicación.
2. *Instalación y ejecución*: descripción de los pasos a seguir para lograr su puesta en marcha, el proceso de instalación.

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

La documentación descrita se encuentra incluida en el CD-ROM que se adjunta y el cual contiene, además, de las distintas librerías de las que se compone la arquitectura la propia aplicación junto con el software necesario para su ejecución.

1.3 Origen del Proyecto

En la actualidad existe un gran interés en el desarrollo de sistemas capaces de procesar documentos de texto automáticamente y de facilitar, en la medida de lo posible el acceso y consulta de los mismos por parte de los usuarios. Grandes cantidades de información de tipo textual están disponibles actualmente en forma más o menos estructurada y en diversos formatos (HTML, PDF, etc.) y la tendencia es a que esos contenidos continúen creciendo. Es más, cualquier organización dispone hoy en día de grandes colecciones de documentos en formato electrónico, hasta el punto de que, en muchos casos, almacenan más información de este tipo que en formato impreso.

Para tratar de simplificar el acceso a esas grandes colecciones de documentos por parte de los usuarios, se han venido desarrollando una serie de algoritmos, técnicas y herramientas dentro del campo de la **Recuperación de Información** [1], como *indexadores*, *clasificadores de texto, generadores de resúmenes, traductores, extractores de palabras clave, clusters*, etc. que ayuden a los usuarios a filtrar y a estructurar la información relevante. Por Recuperación de Información entendemos el proceso de selección de un subconjunto de documentos que se adecuen a las necesidades de información de un usuario, de entre un conjunto más amplio existente en dispositivos de almacenamiento. A continuación se muestran ejemplos de esta clase de métodos:

- La **clasificación de texto** trata la asignación de documentos de texto en formato libre a una o más categorías predefinidas, en base a su contenido. Su principal utilidad surge en entornos donde la organización de documentos es requerida (por ejemplo, en el indexado y filtrado automático de textos).
  Las aproximaciones de clasificación de textos basadas en aprendizaje automático han ganado importancia, llegando a ser las dominantes en la actualidad. Estas aproximaciones aúnan unos buenos niveles de efectividad junto con un considerable ahorro en términos de mano de obra de expertos y cierta independencia del dominio.

- La **extracción de información** es usada para estructurar información específica considerada relevante que se encuentra en documentos de un dominio determinado. En otras palabras, el objetivo de un sistema de extracción de información es encontrar y enlazar la información relevante para un determinado problema, ignorando la extraña e irrelevante.

Muchas de estas técnicas suelen incorporar, en mayor o menor grado, algún tipo de conocimiento lingüístico e incluyen el uso de herramientas de Procesamiento del Lenguaje Natural (PLN) [2] para mejorar su rendimiento y conseguir una mayor flexibilidad.

Sin embargo, la gran variedad de técnicas y métodos disponibles para el acceso a información textual presenta un problema de falta de homogeneidad y de capacidad de integración, lo que dificulta la incorporación de dichas técnicas en aplicaciones prácticas. Por ello resulta conveniente disponer de un marco general en el cual distintas técnicas para el procesamiento de información textual se puedan integrar de forma consistente. De este modo, se podrá simplificar el desarrollo de aplicaciones complejas para al acceso a contenidos textuales en las que se incorporen diferentes técnicas de recuperación de información.

1.4 Objetivos generales del Sistema

La finalidad principal de este proyecto es el desarrollo de un entorno que simplifique el diseño de aplicaciones de acceso a información textual. De un modo más concreto, esto se traduce en los siguientes tres grandes objetivos:

- En primer lugar, se busca realizar un estudio de distintas propuestas disponibles para el procesamiento avanzado de contenidos textuales. Esto supone analizar las aplicaciones típicas (recuperación de documentos, clasificación, etc.) y estudiar los algoritmos empleados y el tipo de información utilizada. Como resultado de este estudio se determinarán las características generales de este tipo de aplicaciones y los aspectos comunes y específicos a las diferentes aproximaciones existentes, en base a los cuales se definirá un esquema general que englobe a este tipo de tareas.

- Como segundo objetivo, se pretende diseñar y construir una arquitectura genérica ("framework") para la integración de servicios de organización, estructuración y acceso a información de tipo textual. Dicha arquitectura proporcionará los componentes básicos y la infraestructura sobre la que construir aplicaciones que integren y/o amplíen los servicios que ésta ofrece. De un modo general, esta arquitectura dará soporte a componentes como:

  - Componentes para la adquisición, captura y estructuración de contenidos documentales.
  - Servicios sobre los documentos individuales, como extracción automática de descriptores o palabras clave, resumen automático, traducción automático y clasificación automática de documentos. Además debería de permitir crear servicios complejos donde se pudiesen integrar diferentes servicios sobre documentos individuales, como por ejemplo permitir realizar un resumen sobre una traducción que ya se haya realizado.

  - Servicios sobre grupos de documentos o de "agregación de documentos", como son el proceso de indexación y agrupamiento de textos similares también llamado clustering.
  - Soporte para la agrupación de servicios, es decir, el poder indicar de un modo sencillo una cadena de los servicios anteriores sobre documentos que se quiere realizar hasta la obtención de un resultado.
  - Servicios de presentación de los documentos. Se van a distinguir dos tipos de presentaciones. Por un lado la presentación de cualquiera de los documentos originales y documentos generados tras el paso de un servicio sobre un documento individual; y la presentación de la vista de los grupos de documentos obtenidos en forma de lista tras el paso de un servicio sobre grupos de documentos.

- Por último, se plantea construir un pequeño ejemplo de aplicación concreto basado en la arquitectura diseñada, que integre diversos componentes con el propósito de mostrar su funcionamiento y su utilidad práctica.

Dada la entidad de este proyecto y la planificación temporal que se ha previsto, no se puede pretender realizar un estudio exhaustivo de todos los posibles tipos de aplicaciones de procesamiento documental propuestos actualmente. Nuestra aproximación se basará en un estudio de tareas prototipo (clasificación, búsqueda, etc.) tomando como punto de partida aplicaciones y algoritmos concretos para los cuales existan implementaciones disponibles y fácilmente accesibles. De tal modo que, en base al análisis de esos casos, podamos crear un esquema genérico que pueda integrar esas tareas tipo de una forma homogénea.

1.5 Estudio previo de herramientas

Para desarrollar la arquitectura genérica que proponemos se partió de un estudio inicial sobre herramientas de recuperación información, buscando las características de éstas para analizar y finalmente obtener una arquitectura que permitiese el desarrollo de una amplia variedad de aplicaciones sobre ella. A medida que se iba indagando en el tema, fueron apareciendo más y más herramientas que proporcionaron datos de interés para este estudio, algunas de las más relevantes se enumeran a continuación:

- Lucene[6]:Es una novedosa herramienta que permite tanto la indexación como la búsqueda de documentos. A pesar de solo trabajar con texto, posee complementos añadidos que permiten indexar documentos de Word, ficheros PDF, XML o páginas HTML. Creada e implementada completamente en Java, se trata de una API flexible, muy potente y fácil de usar, a través de la cual se pueden añadir, con pocos esfuerzos de programación, capacidades de indexación y búsqueda a cualquier sistema que se esté desarrollando.(ver el apéndice A, página 201)
- Classifier4j: Es una librería de clasificación de textos implementado en Java que incluye un resumidor de textos y un clasificador Bayesiano. Este API es muy sencillo de usar y permite incorporar tanto el resumidor como el

clasificador Bayesiano en cualquier aplicación con muy pocas líneas de código.(ver el apéndice)

- LexRank[8]: Es un método desarrollado por la universidad de Michigan para computar unidades textuales de Procesamiento de lenguajes naturales, aplicado sobre todo a resúmenes sobre textos. Permite extraer las frases más importantes de un documento o de un conjunto de documentos, identificando términos típicos presentes en el documento. Si bien este método parece bastante eficaz, la falta de disponibilidad de la herramienta, no hizo posible su instalación.
- Lemur: Es un framework desarrollado con la colaboración del departamento de informática de la Universidad de Massachussets y la Universidad de Carnegie Mellon para la recuperación de información y diseñado para facilitar la investigación. Incluye soporte para tareas como las búsquedas *ad-hoc* y permite resúmenes, filtración y clasificación. También permite indexación de las direcciones de las bases de datos. Está escrito en C y C++ y está diseñado para funcionar bajo Unix aunque desde hace muy poco también para Windows.
- Weka: Implementa numerosos algoritmos de aprendizaje y múltiples herramientas para transformar las bases de datos y realizar un exhaustivo análisis, como son tareas de clasificación, regresión, clustering, asociación y visualización. Weka está diseñado como una herramienta orientada a la extensibilidad por lo se ha incluido por sus facilidades para el desarrollo de clasificadores y agrupadores (clústeres) de texto. (ver el apéndice C, página 208)

Otros sistemas menos relevantes aportaron también datos de interés a nuestra arquitectura como pueden ser *Kea*, que está basado casi totalmente en *Weka*, e *Indri* que es la primera versión de *Lemur* y posee muchas menos funcionalidades que ésta. Además de estas herramientas se encontraron otras que no se emplearon como por ejemplo:

- Pertinence: Es una aplicación de producción de resúmenes automáticos que tiene en cuenta la especificidad del texto y su temática basándose exclusivamente en técnicas de análisis lingüístico y semántico. El tratamiento de información textual es multilingüe, soportando 15 idiomas. Posee una versión demo accesible desde la url: http://www.pertinence.net/ps/index.jsp?ui.lang=fr. Su uso tuvo que ser descartado por ser una herramienta de pago.
- OTS: Es una herramienta open source para resumir textos. El programa lee un texto y decide cuales son las frases importantes y cuales no. Esta librería es multilingüe y trabaja con encoding UTF-8, por lo que permite resumir textos en inglés, alemán, español, etc. En caso de querer soportar más idiomas solo es necesario editar un fichero XML con la lista de reglas. Su uso se tuvo que desechar ya que su instalación en su momento fue imposible por la deficiente documentación proporcionada.

De este primer estudio se obtienen las bases para comenzar a desarrollar una arquitectura con el objetivo de poder implementar sobre ella aplicaciones para la gestión de documentos textuales que incorpore tareas tales como un servicio de resúmenes, de extracción de palabras clave y un indexador.

1.6 Descripción del Sistema

En este apartado se mencionan los datos técnicos esenciales del proyecto desarrollado así como los recursos hardware y software necesarios para la consecución del sistema. Además se explica cual fue la solución adoptada para la realización tanto de la arquitectura genérica como la implementación de la aplicación de ejemplo que se realizó con la misma.

1.6.1 Solución Adoptada

Para poder explicar de un modo riguroso la solución que se desarrolló a la hora de realizar esta aplicación es necesario hacer una distinción; por un lado la arquitectura y por el otro la implementación de un caso concreto de esa arquitectura. Por ello a continuación se detallan cada uno de los aspectos de las distintas partes.

1.6.1.1 Descripción general de la arquitectura

En el gráfico que se muestra a continuación, se representan los componentes del sistema desarrollado en la arquitectura. En él se pueden observar las diversas tareas a las que se pretenden dar soporte con la arquitectura propuesta.

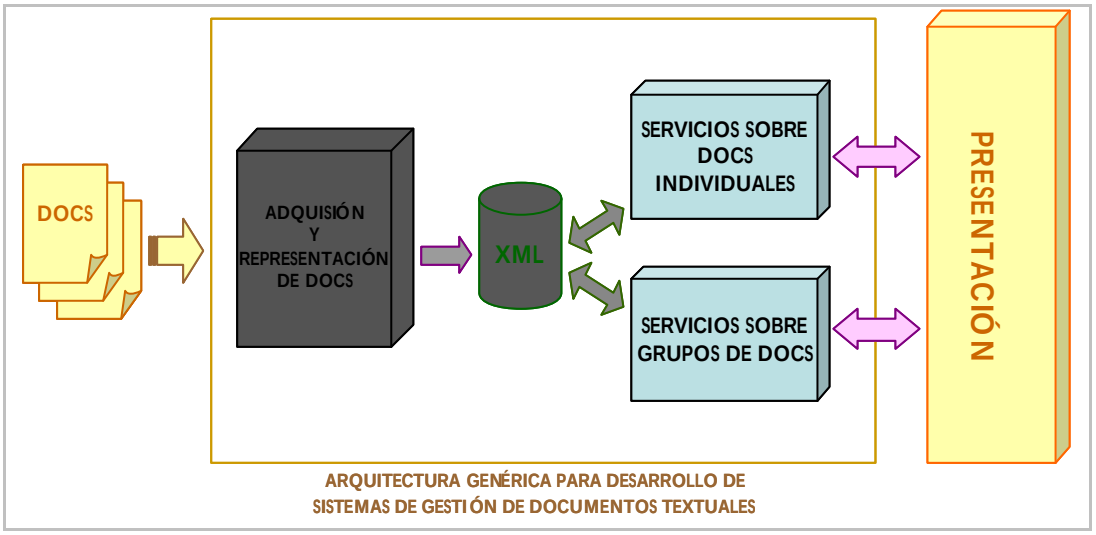


Figura 1: Arquitectura Genérica

La arquitectura que implementa este proyecto incorpora un subsistema de adquisición y representación de documentos que permite guardar toda la información extraída en un formato estándar común para las posibles aplicaciones a desarrollar. En nuestro caso se ha optado por usar XML [3] como formato base por unos motivos bien definidos:

- XML, es el estándar de *Extensible Markup Language*, que no es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes.
- Es una arquitectura más abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados.
- Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML.
- Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.
- Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido en XML consigue que la estructura de la información resulte más accesible.

Antes de seguir con la descripción general de la arquitectura es necesario indicar que el documento obtenido mediante un documento original y tras extraer toda la información relevante pasa a llamarse **documento base** siguiendo nomenclatura de la arquitectura propuesta.

A partir de esta representación interna de los documentos a gestionar, será necesario distinguir los servicios que se pueden realizar sobre esos documentos de un modo individual y los que se pueden ofrecer sobre grupos de documentos.

La arquitectura implementada distingue por tanto entre esos dos tipos de elementos. A continuación se detallan cuales son los componentes incluidos de cada uno de ellos.

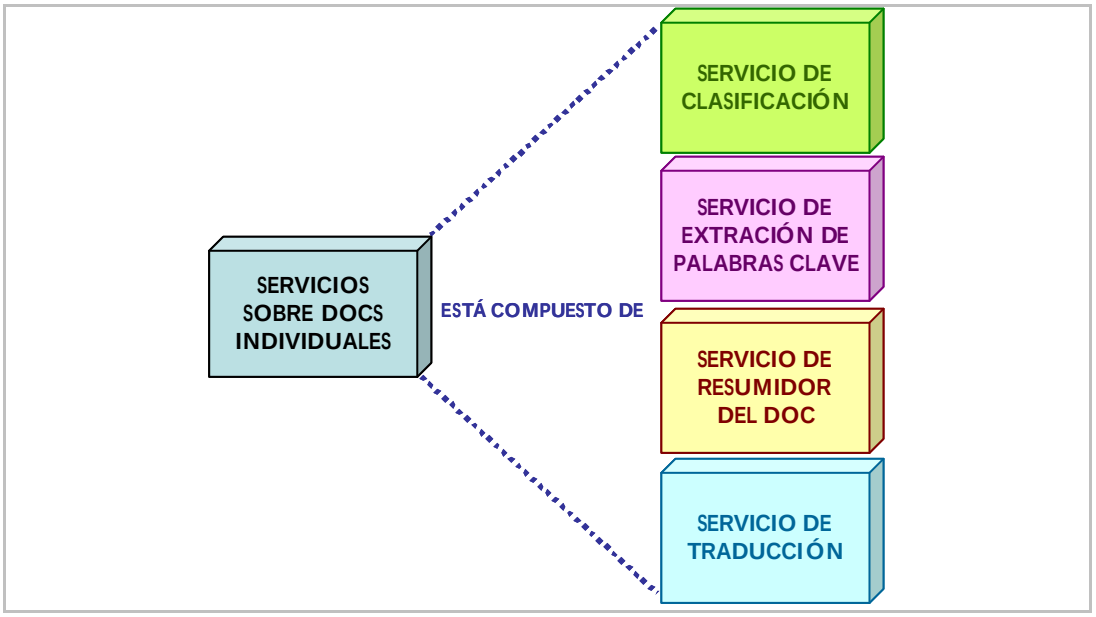


Figura 2: Servicios sobre Documentos Individuales

En una primera aproximación, se pretende dar soporte a estos tipos de servicios sobre documentos individuales. De manera más detallada cada uno de estos servicios proporciona al sistema una funcionalidad específica:

- Los servicios de extracción de palabras clave permiten obtener las palabras clave de un documento.
- Los servicios de resumidor de un documento permiten obtener un resumen asociado al mismo.
- Los servicios de traducción permiten obtener la traducción de un documento a cualquier idioma.
- Los servicios de clasificación de un documento permiten clasificar al mismo dentro de unas categorías preestablecidas.

Debido a que se optó por desarrollar una aproximación basada en un diseño orientado a objetos, es necesario indicar que en las aplicaciones desarrolladas sobre la arquitectura existirá una clase responsable de implementar cada uno de los servicios.

Cada uno de estos servicios una vez aplicados a un documento genera lo que se llamará a lo largo de este manual técnico un **elemento de visión**. De este modo cuando se realice la extracción de palabras clave sobre un documento base, el resultado obtenido será una visión de extracción aplicado a ese documento. Del mismo modo si se procediese a realizar a continuación una traducción de la visión de extracción se obtendría una visión de traducción de la extracción. Así tantas veces como sea necesario para la aplicación concreta. La obtención de estas visiones es lo que denominaremos un **documento decorado** o un elemento de decoración.

Por otro lado se encuentran los servicios proporcionados sobre grupos de documentos, que siguiendo con nuestro primer acercamiento, pretende dar soporte a los que a continuación se detallan:

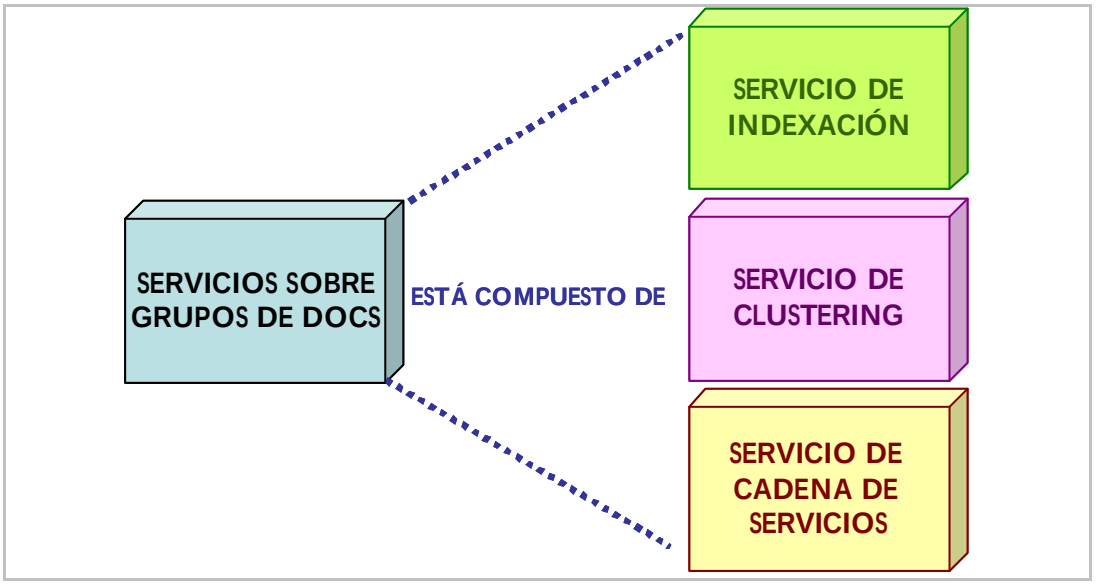


Figura 3: Servicios sobre Grupo de Documentos

- El servicio de indexación permite al sistema indexar los documentos para posteriormente poder realizar búsquedas sobre ellos.

- El servicio de clustering permite al sistema agrupar los documentos para identificar todos aquellos cuyo contenido trata del mismo tema.
- El servicio de cadenas de servicios permite al sistema establecer cuales van a ser los pasos a seguir y el orden en el que se van a desarrollar sobre servicios de grupos de documentos. Por ejemplo se podrá indicar que se va a realizar una indexación a través de uno o varios términos de consulta y que sobre el resultado se aplicará el clustering. Por lo tanto el resultado obtenido será un conjunto de documentos relevantes para esa consulta organizados en grupos de acuerdo a su temática.

Para terminar, se encuentran los servicios de presentación, proporcionados tanto para los documentos base, como para sus decoraciones, que se estructuran como se detallan a continuación:

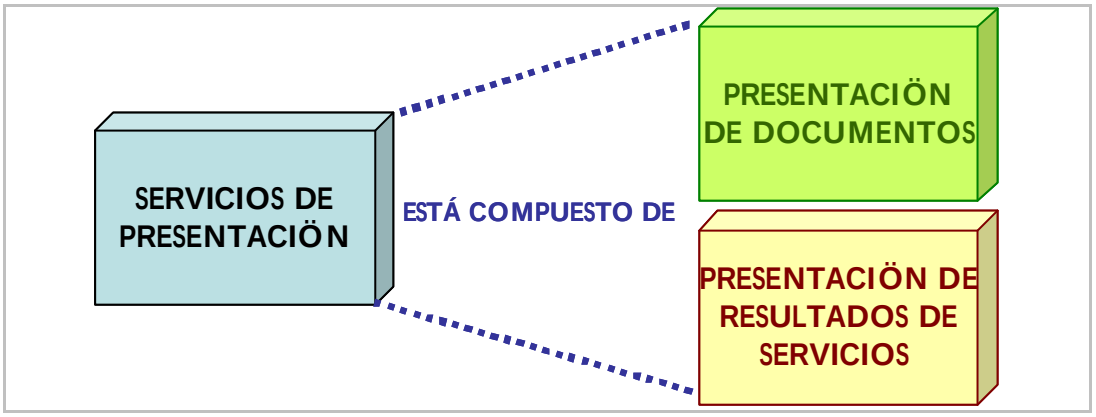


Figura 4: Servicios de presentación

En este primer desarrollo, se decidió dar soporte a un tipo de presentación bien definido como es el orientado a Web, basado en la visualización de fragmentos de documentos HTML integrados en páginas JSP.

La idea es que la propia arquitectura lleve incrustado una clase que implemente una visualización por defecto tanto de la presentación de documentos como de los resultados de los servicios. De este modo el desarrollador que use esta arquitectura, siempre tendrá este tipo de presentación disponible. En caso contrario solo tendrá que redefinirlas.

1.6.1.2 Descripción de la implementación de la arquitectura

Lo que se pretende implementando una aplicación que use la arquitectura propuesta es proporcionar un ejemplo del uso de la misma.

La aplicación que implementa este proyecto incorpora diversos servicios como son los que a continuación se indican y que proporcionan al sistema una funcionalidad específica:

- Un algoritmo de clasificación

- un sistema de extracción de palabras clave
- un algoritmo de resúmenes
- un sistema de traducción

- El **motor de búsqueda**, se encarga de capturar, analizar y obtener la representación adecuada para la consulta introducida por el usuario. Una vez procesada la consulta, recorre el índice de los documentos identificando aquellos que son relevantes. Es necesario indicar que los documentos sobre los que se realizará el proceso de construcción de la representación interna de la indexación pueden proceder de diversos documentos. Así se podrán tener múltiples indexadores: un indexador podría tratar solo documentos base; otro podría tratar documentos que proceden de una visión, etc.

- El **algoritmo de clustering o agrupamiento**, distribuye los documentos en una serie de clusters o grupos, teniendo en cuenta que todos aquellos documentos que pertenezcan a un mismo grupo están relacionados por su contenido. Es necesario indicar que los documentos sobre los que se realizará el proceso de construcción de la representación interna del agrupamiento pueden proceder de diversos documentos. Así se podrán tener múltiples agrupadores: un clustering podría tratar solo documentos base; otro podría tratar documentos que proceden de una visión, etc.

- Pero además, arquitectura incorpora la funcionalidad de poder indicar en que orden se quieren realizar la cadena de servicio. Por ejemplo una cadena podría ser que el primer paso fuese indexar documentos base y sobre los resultados obtenidos indexarlos de nuevo pero esta vez los documentos serán visiones de los documentos base.

La aplicación está basada en una arquitectura cliente-servidor con las siguientes pautas de actuación:

- Una vez que un usuario ejecuta un navegador Web en su ordenador, éste actúa como cliente realizando peticiones al servidor en el que reside la aplicación. A través de la interfaz de usuario introduce su consulta, la cual es enviada al servidor.
- El servidor escucha las peticiones realizada por el cliente y lleva a cabo todas aquellas operaciones que sean necesarias para atender dichas peticiones. En este caso, las operaciones a realizar implican la identificación de los documentos relevantes a la consulta recibida (obtenidas por clustering, por indexación o por cadena de servicios) y todas aquellas operaciones que se pudieran ejecutar sobre el documento de modo individual. Una vez obtenido el resultado, se envía la respuesta al cliente.
- De nuevo en el cliente, el navegador interpreta las respuestas enviadas por el servidor mostrando los resultados al usuario.

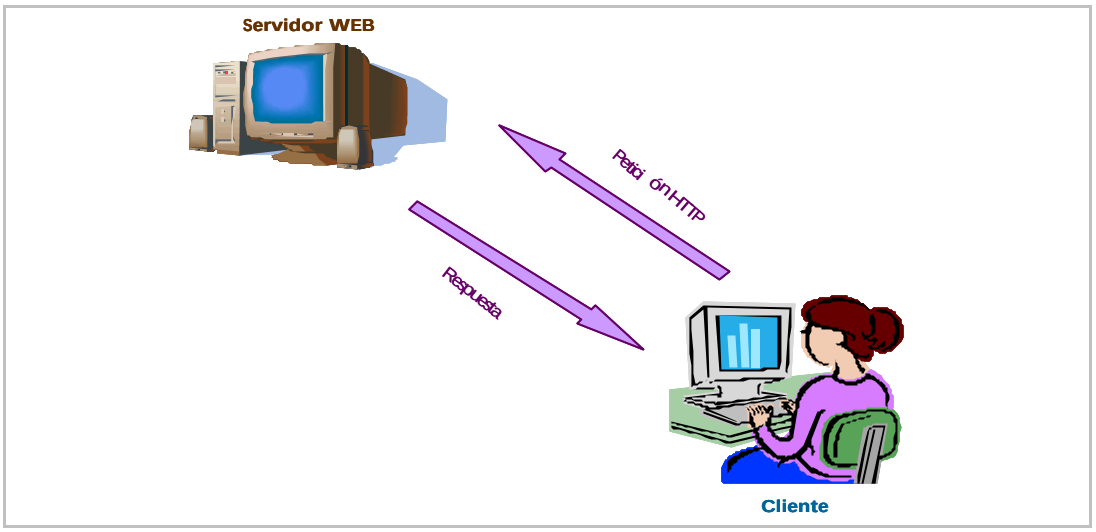


Figura 5: Arquitectura Implementación Concreta (cliente-servidor)

1.6.2 Datos técnicos del proyecto

Para el desarrollo de esta aplicación se necesitó el uso de diferentes tecnologías y lenguajes de programación. A continuación se muestra un listado de las herramientas y tecnologías utilizadas para llevar a cabo las distintas fases de desarrollo del proyecto (planificación, análisis, diseño, implementación y pruebas tanto de la arquitectura desarrollada como de la implementación):

- **Planificación**:
  La forma de representar la estimación temporal y de costes de las etapas de desarrollo del proyecto, se realizó por medio del uso de los diagramas de Gantt utilizando el programa "***Microsoft Project 2003***".

- **Análisis y diseño:**
  El análisis y diseño se realizó mediante el uso del Lenguaje Unificado de Modelado UML empleando como herramienta el "*Rational Rose 2000 Enterprise Edition*".

- **Implementación:**
  Para la implementación, se va a distinguir entre las herramientas usadas para la realización de la arquitectura y las herramientas usadas para la realización de la aplicación concreta.

  Para la implementación de la arquitectura, se usó:
  - ***Java***:
    - ***MD5:*** librería que permite crear identificadores hash haciendo uso de uno de los algoritmos de reducción criptográficos
    - ***regexp:*** clase que implementa expresiones regulares

  - ***XML***: usado para la creación de los documentos que genera la arquitectura

- ***XSLT:*** usado para la generación de las diversas presentaciones que forman las implementaciones por defecto de la arquitectura.

La implementación de la aplicación se llevó a cabo integrando diversos componentes, como pueden ser:

- ***Lucene***: Herramienta implementada completamente en Java, flexible, muy potente y fácil de usar, a través de la cual se pueden añadir con pocos esfuerzos de programación, capacidades de indexación y búsqueda a cualquier sistema que se esté desarrollando. Es funcional bajo cualquier entorno.
- ***Google***: Buscador con herramientas de traducción. Es funcional bajo cualquier entorno
- ***Classifier4j***: librería implementada completamente en Java, que permite clasificar texto así como realizar resúmenes. Es funcional bajo cualquier entorno
- ***Weka***: librería implementada completamente en Java, que permite realizar clasificación y agrupamiento de texto. Es funcional bajo cualquier entorno
- ***Erial***: herramienta implementada en Perl, que permite realizar extracción de palabras clave. Es funcional bajo un entorno Linux.

También se utilizó:

- ***JSP*** (Java Server Pages) y ***Servlets***: usado para la implementación de las páginas de servidor.
- ***Javascript***: usado para la programación de aspectos relacionados con la interacción con el usuario.
- ***HTML***: usado para la creación de las páginas cliente de la aplicación.
- ***Flash***.

1.6.3 Entornos Hardware y Software

El detalle de los recursos necesarios para el desarrollo del proyecto se encuentra en la tabla que se muestra a continuación:

REQUISITOS SOFTWARE
Sistema Operativo Debian
Servidor de páginas Tomcat
Navegador Web Firefox versión 1.7.10
Java versión 1.4.2.09
Eclipse Project, versión 3.0 para el desarrollo de la aplicación en java
Sistema Operativo Windows XP
Microsoft Project 2003, herramienta utilizada para la realización de la planificación y estimación de costes

Rational Rose 2000 Enterprise Edition: herramienta empleada para la realización del análisis y diseño de la documentación de la aplicación
Microsoft Office 2003: utilizado para la realización de la documentación
Macromedia DreamWeaver MX 2004: utilizado para el desarrollo de las páginas JSP y HTML.
Macromedia Fireworks MX 2004: utilizado para la maquetación de la interfaz de usuario.
Macromedia Flash MX 2004: utilizado para la realización de la pagina de carga inicial

REQUISITOS HARDWARE
PC con Procesador Pentium a 400Mhz
Unidad de CD-ROM
Conexión a Internet

2. Desarrollo del Proyecto

En este apartado se describen los diversos aspectos del desarrollo del proyecto como son: la metodología empleada para el análisis y el diseño, la solución adoptada para la implementación, la planificación llevada a cabo y las especificaciones de los requisitos del sistema.

2.1 Lenguaje de Modelado.

Para poder llevar a cabo el análisis y diseño del sistema, se empleó el Lenguaje Unificado de Modelado (UML)[4]. UML es el sucesor de los diversos métodos de análisis y diseño orientados a objetos que surgió a finales de la década de los 80 y principios de la siguiente, unificando de este modo los métodos de *Rumbaugh*, *Jacobson* y *Booch*.

Hoy en día, UML ("*Unified Markup Language*") [4] esta consolidado como el lenguaje estándar en el análisis y diseño de sistemas de computo. Mediante UML es posible establecer la serie de requerimientos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código.

En otros términos, así como en la construcción de un edificio se realizan planos previo a su construcción, en Software se deben realizar diseños en UML previos codificación de un sistema, ahora bien, aunque UML es un lenguaje, éste posee más características visuales que programáticas.

Cuanto más complejo es el sistema que se desea crear, más beneficios presenta el uso de UML. Las razones de esto son evidentes, sin embargo, existen dos puntos claves: El primero se debe a que mediante un plano/visión global resulta más fácil detectar las dependencias y dificultades implícitas del sistema, y la segunda razón radica en que los cambios en una etapa inicial (Análisis) resultan más fáciles de realizar que en una etapa final de un sistema como lo seria la fase intensiva de codificación.

Además, UML proporciona una forma estándar de escribir tanto las cosas conceptuales como las concretas, manteniendo cierta independencia entre el modelado y el lenguaje de programación específico, y los componentes software reutilizables. Además, es necesario recalcar que UML es una notación, no un estándar de proceso y que una de las características más relevantes es su capacidad para absorber la nueva semántica sin romper su lógica interna.

Puesto que UML es empleado en el análisis para sistemas de mediana-alta complejidad, era de esperarse que su base radique en otro paradigma empleado en diseños de sistemas de alto nivel que es la orientación a objetos.

También es necesario indicar que una aplicación Web como la que se desarrolló como ejemplo de funcionamiento de la arquitectura, puede ser modelada mediante UML. Sin embargo, el modelo del análisis y diseño presenta algunas dificultades y limitaciones al tratar de incluirlas, ya que non existen técnicas que capturen la semántica de elementos como son los enlaces.

Por lo tanto se recurrió al uso de las recomendaciones de Conallen para modelar arquitecturas Web con UML (*la Web Extension Application, WAE 1998*) [9] que permite rentabilizar toda la gramática interna de UML para modelar aplicaciones con elementos específicos de la arquitectura de un entorno Web. Dicha extensión presenta la posibilidad de extender una semántica adicional al UML tradicional a través de un mecanismo de extensión formal, el cual nos permite definir estereotipos, valores etiquetados y restricciones que pueden ser aplicados a elementos del modelo para permitir elementos específicos Web, utilizándolo tanto en el análisis como en el diseño.

Un estereotipo es un mecanismo que nos permite definir una nueva semántica para modelar elementos. Los valores etiquetados son pares de llave-valor que nos permite etiquetar cualquier valor en un elemento del modelo. Las restricciones son reglas que definen la forma correcta del modelo.

Las páginas Web se representan una a una con componentes en UML, los cuales son partes físicas de un sistema. La vista de implementación de un modelo describe los componentes del sistema y sus relaciones. En una aplicación Web, éstos describen cada una de las páginas y sus relaciones con otras.

Cada página Web se representa con una clase en la vista de modelos de diseño, e sus relaciones con otras páginas, representan los hiperenlaces. Pero esta abstracción se dificulta cuando una determinada página puede realizar algo en el lado del servidor y otra cosa distinta en el lado del cliente. Por eso, los aspectos de una página en el lado del servidor pueden ser modelados con una clase y los del lado del cliente con otra distinta, distinguiendo entre las dos con un mecanismo de extensión para definir los estereotipos e iconos de cada uno.

Para realizar el análisis y diseño de la aplicación, se utilizó un soporte idóneo para UML, como es "*Rational Rose 2000 Enterprise Edition*".

2.2 Ciclo de Vida

El ciclo de vida ofrece un procedimiento común a seguir para desarrollar un sistema computacional que pueda servir para orientar hasta que finalice dicho proyecto.

Los objetivos del ciclo de vida son:

- Definir las actividades que se deben llevar a cabo en un proyecto de desarrollo de un sistema, logrando cierta congruencia.
- Proporcionar puntos de control y de revisión administrativos, de tal forma que se pueda evaluar el sistema.

Los modelos de ciclo de vida proponen un conjunto de pasos que abarcan métodos, herramientas y procedimientos. La elección de uno de ellos depende exclusivamente del tipo de proyecto que se está efectuando.

Para el desarrollo de este proyecto se ha elegido como proceso de desarrollo el denominado **RUP.** RUP es uno de los procesos más generales del los existentes actualmente, ya que en realidad esta pensado para adaptarse a cualquier proyecto, y no tan solo de software.

Un proyecto realizado siguiendo RUP se divide en cuatro fases:

- Intercepción (puesta en marcha)
- Elaboración (definición, análisis, diseño)
- Construcción (implementación)
- Transición (fin del proyecto y puesta en producción)

En cada fase se ejecutó una o varias iteraciones, y dentro de cada una de ellas siguió un modelo de cascada para los flujos de trabajo que requieren las actividades anteriormente citadas.

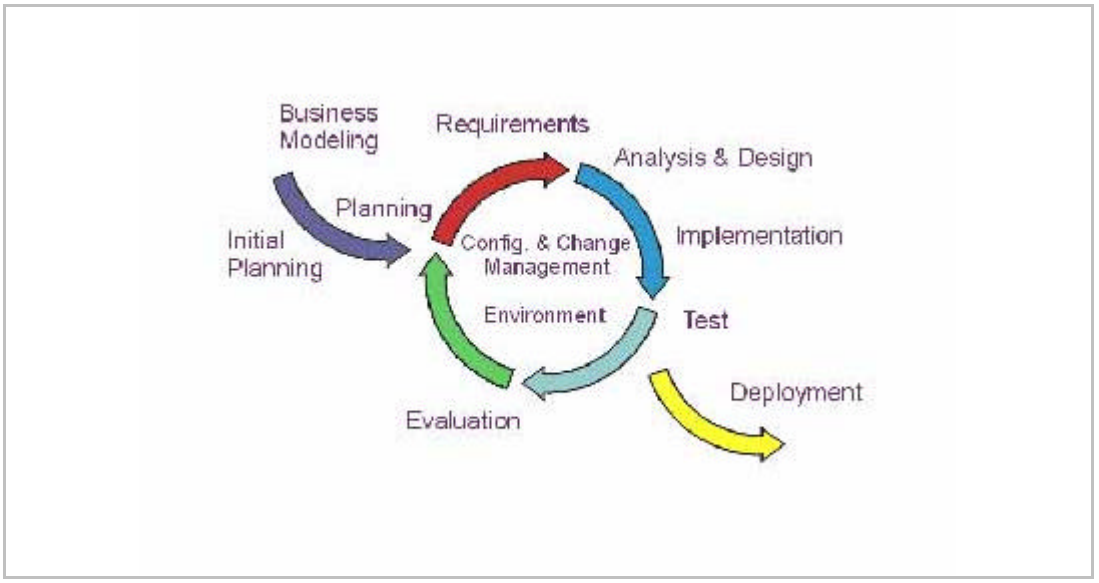


Figura 6: Flujos de trabajo de RUP

RUP se basa en casos de uso para describir lo que se espera del software y esta muy orientado a la arquitectura del sistema, documentándose lo mejor posible, basándose en UML (*Unified Modeling Language*) como herramienta principal.

2.3 Planificación

El objetivo, de esta fase es establecer los plazos para la finalización de cada unas de las etapas en las que se dividen el desarrollo del sistema. También, se realiza una

estimación de los costes del proyecto, así como de los recursos humanos que fueron necesarios.

La forma de representar la estimación temporal de las etapas del proyecto, fue mediante los diagramas de **Gantt**, ya que se consideran claros, concisos y de fácil comprensión, además de proporcionar una visión global de todo el ciclo de desarrollo. Dichos diagramas se realizaron mediante el programa "*Microsoft Project 2003*".

Para calcular el coste estimado para el proyecto, se tuvo en cuenta las horas trabajadas durante toda la semana.

ESTIMACIONES	TIEMPO
Número de semanas	18 semanas
Horas estimadas por semana	48 horas semanales
Total de horas	864 horas

FASES	PRECIOS
Fase de análisis	18 €/ hora
Fase de diseño	18 € /hora
Otras fases	15 €/ hora

FASES	TIEMPO	COSTE
Fase de análisis y diseño	9 semanas	7776 €
Resto de las fases	11 semana	7920 €
Total da aplicación	18 semanas	15696 €

COSTES SOFTWARE	PRECIOS
Microsoft Windows XP Professional	129 €
Macromedia Studio MX 2004	299 €
Microsoft Project 2003	868 €
Microsoft Word 2003	338 €
Total:	1634 €

El total de la aplicación asciende a: **17330 €** sin incluir los gastos del Hardware.

2.3.1 Planificación Previa

Inicialmente se propuso las siguientes fases en el desarrollo del proyecto:

- Estudio previo de aplicaciones tipo
  - Estudio del problema propuesto
  - Estudio del sistema
- Desarrollo de la plataforma base
  - Análisis de la plataforma base
  - Diseño de la plataforma base
  - Diseño de los componentes básicos
  - Implementación de la plataforma base
  - Pruebas de la plataforma base
- Desarrollo de aplicación concreta
  - Diseño e implementación de la aplicación ejemplo
  - Pruebas de la aplicación ejemplo
- Documentación
  - Documentación del análisis de la plataforma base
  - Documentación del diseño de la plataforma base
  - Manual técnico de la plataforma base
  - Manual de usuario de la plataforma base
  - Documentación del diseño de la aplicación ejemplo

Como se puede observar, en el siguiente gráfico, el desarrollo del proyecto se estimó en un principio que se iba necesitar 77 días, en los que la carga de trabajo semanal fuese de 37 horas. Como se puede ver en la figura, la suma de los días indicados en cada una de las tareas no corresponde con la suma final de los 77 días. Esto se debe a que ciertas tareas se iban a ejecutar de forma paralela.

Estimación temporal:

	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto**	**77 días**	**mié 01/06/05**	**lun 26/09/05**
2	**Estudio previo de aplicaciones tipo**	**12 días**	**mié 01/06/05**	**lun 20/06/05**
3	Estudio del problema propuesto	12 días	mié 01/06/05	lun 20/06/05
4	Estudio del sistema	10 días	mié 01/06/05	jue 16/06/05
5	**Desarrollo de la plataforma base**	**49 días**	**vie 17/06/05**	**mar 30/08/05**
6	Análisis de la plataforma base	13 días	vie 17/06/05	jue 07/07/05
7	Diseño de la plataforma base	10 días	vie 08/07/05	jue 21/07/05
8	Diseño de los componentes básicos	10 días	lun 18/07/05	lun 01/08/05
9	Implementación de la plataforma base	22 días	mié 27/07/05	vie 26/08/05
10	Pruebas de la plataforma base	8 días	vie 19/08/05	mar 30/08/05
11	**Desarrollo de aplicación concreta**	**16 días**	**mié 31/08/05**	**jue 22/09/05**
12	Diseño e implementación de la aplicació	12 días	mié 31/08/05	vie 16/09/05
13	Pruebas de la aplicación ejemplo	7 días	mié 14/09/05	jue 22/09/05
14	**Documentación**	**67 días**	**vie 17/06/05**	**lun 26/09/05**
15	Documentación del análisis de la plataf	20 días	vie 17/06/05	lun 18/07/05
16	Documentación del diseño de la platafo	20 días	mar 19/07/05	mié 17/08/05
17	Manual técnico de la plataforma base	5 días	mié 31/08/05	mié 07/09/05
18	Manual de usuario de la plataforma bas	5 días	jue 08/09/05	mié 14/09/05
19	Documentación del diseño de la aplicac	8 días	jue 15/09/05	lun 26/09/05

Figura 7: Estimación Temporal Planificación Previa

Diagrama de Gantt:

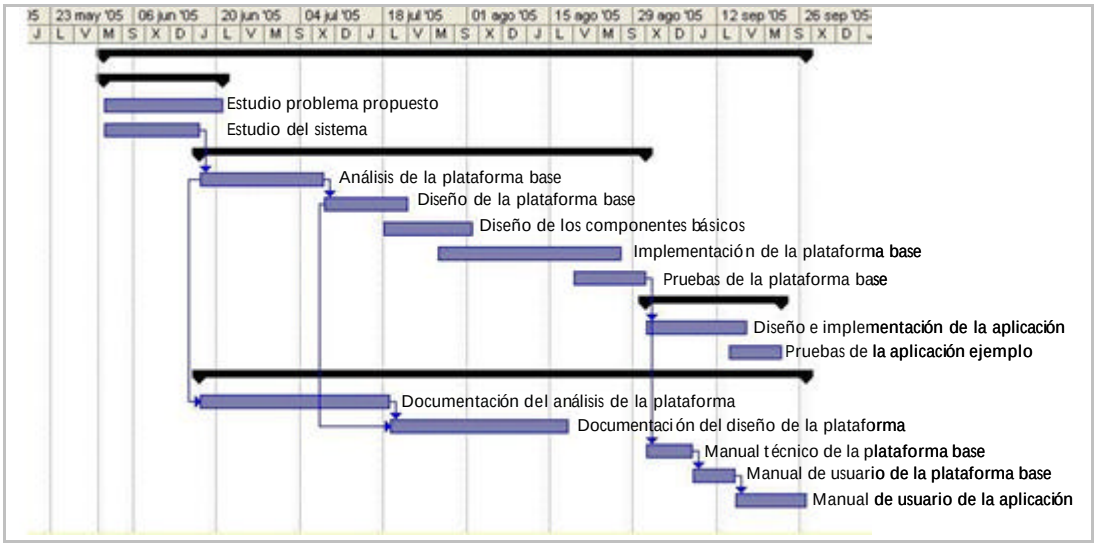


Figura 8: Diagrama de Gantt Panificación Previa

2.3.2 Planificación Real

Las fases llevadas a cabo para el desarrollo del sistema siguieron siendo las mismas que las que se propusieron inicialmente:

Estimación temporal:

	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto**	**87,5 días**	**mar 07/06/05**	**mié 28/09/05**
2	**Estudio previo de aplicaciones tipo**	**19,5 días**	**mar 07/06/05**	**lun 04/07/05**
3	Estudio del problema propuesto	12 días	mar 07/06/05	mié 22/06/05
4	Estudio del sistema	10 días	lun 20/06/05	lun 04/07/05
5	**Desarrollo de la plataforma base**	**56,25 días**	**lun 04/07/05**	**mié 14/09/05**
6	Análisis de la plataforma base	15 días	lun 04/07/05	jue 21/07/05
7	Diseño de la plataforma base	10 días	vie 22/07/05	jue 04/08/05
8	Diseño de los componentes básicos	11 días	mar 02/08/05	mar 16/08/05
9	Implementación de la plataforma base	25 días	vie 12/08/05	mié 14/09/05
10	Pruebas de la plataforma base	8 días	vie 19/08/05	lun 29/08/05
11	**Desarrollo de aplicación concreta**	**24 días**	**lun 29/08/05**	**mié 28/09/05**
12	Diseño e implementación de la aplicació	12 días	lun 29/08/05	mar 13/09/05
13	Pruebas de la aplicación ejemplo	10 días	vie 16/09/05	mié 28/09/05
14	**Documentación**	**55,75 días**	**vie 15/07/05**	**sáb 24/09/05**
15	Documentación del análisis de la plataf	10 días	vie 15/07/05	jue 28/07/05
16	Documentación del diseño de la platafo	10 días	lun 01/08/05	vie 12/08/05
17	Manual técnico de la plataforma base	8 días	lun 29/08/05	jue 08/09/05
18	Manual de usuario de la plataforma bas	5 días	jue 08/09/05	mié 14/09/05
19	Documentación del diseño de la aplicac	8 días	jue 15/09/05	sáb 24/09/05

Figura 9: Estimación temporal Planificación Real

Diagrama de Gantt:

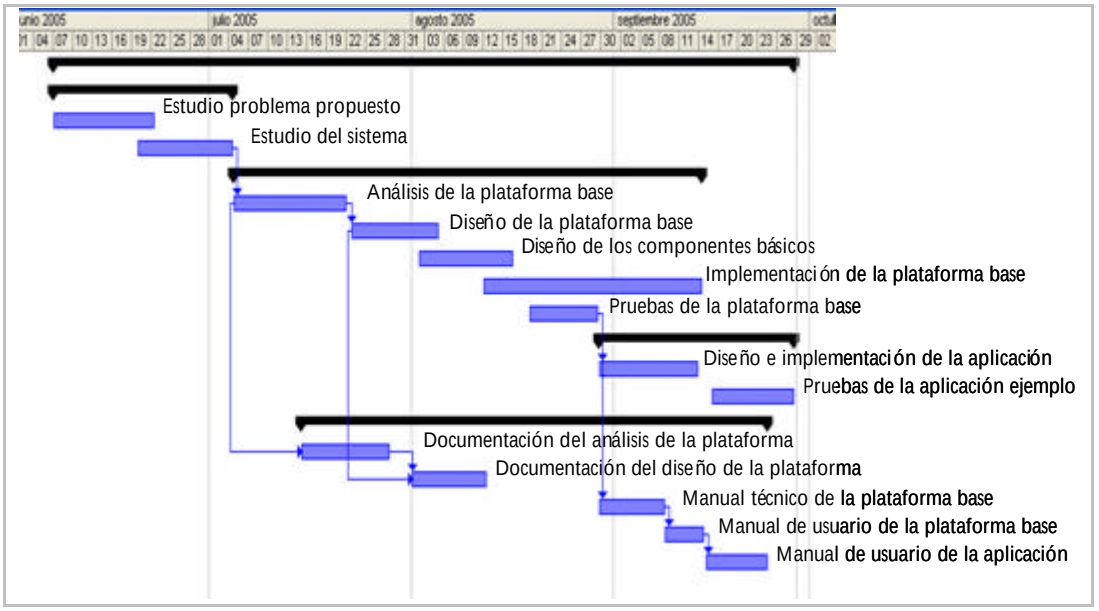


Figura 10: Diagrama de Gantt Planificación Real

Hay que destacar que se ha pasado de una estimación inicial de 37 horas semanales, a tener que realizar un promedio de 48 horas semanales. Esto es, debido a la complejidad del sistema, se tuvo que incrementar los días laborables de 5 a 6 días para poder mantener los plazos previstos.

3. Análisis y Diseño de la Arquitectura

Para desarrollar la arquitectura se partió de un estudio inicial sobre herramientas de indexación basándose en Lucene, Classifier4j y de herramientas de extracción de palabra clave como Erial, buscando las características de éstos para analizar y finalmente obtener una arquitectura genérica que permitiese el mayor número de aplicaciones sobre ella.

3.1 Análisis de la arquitectura

A continuación se procede a detallar por un lado las funcionalidades iniciales de la arquitectura, el diagrama de clases y finalmente los diagramas de casos de uso asociados.

3.1.1 Funcionalidades de la arquitectura

Las funcionalidades que se identificaron en el sistema se presentan en la siguiente tabla:

FUNCIONALIDADES
1. Añadir un documento al sistema
2. Añadir un directorio al sistema
3. Acceder a servicios y presentar los documentos base en base a parámetros
4. Acceder a servicios y presentar las visiones en base a parámetros
5. Crear un documento base a partir de un documento original
6. Crear las visiones asociadas a un documento base
7. Crear las visiones asociadas a otras visiones
8. Crear los servicios asociados a grupos de documentos
9. Crear la visualización de un documento base
10. Crear la visualización de las visiones
11. Presentar un documento base
12. Presentar una visión de un documento base
13. Presentar una visión de visiones

14. Eliminar un documento base del sistema
15. Eliminar las visiones asociadas a un documento base
16. Eliminar las visiones asociadas a las visiones
17. Eliminar los servicios asociados a un grupo de documentos
18. Eliminar la visualización de un documento base
19. Eliminar la visualización de las visiones
20. Modificar los servicios asociados a un grupo de documentos

3.1.2 Diagrama de Casos de uso general

Los diagramas de casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista del usuario.

Un caso de uso es una manera específica de utilizar el sistema. Es la funcionalidad del sistema que se desencadena en respuesta a la estimulación de un actor externo. La siguiente figura muestra el diagrama de casos de uso que denota el comportamiento del sistema.

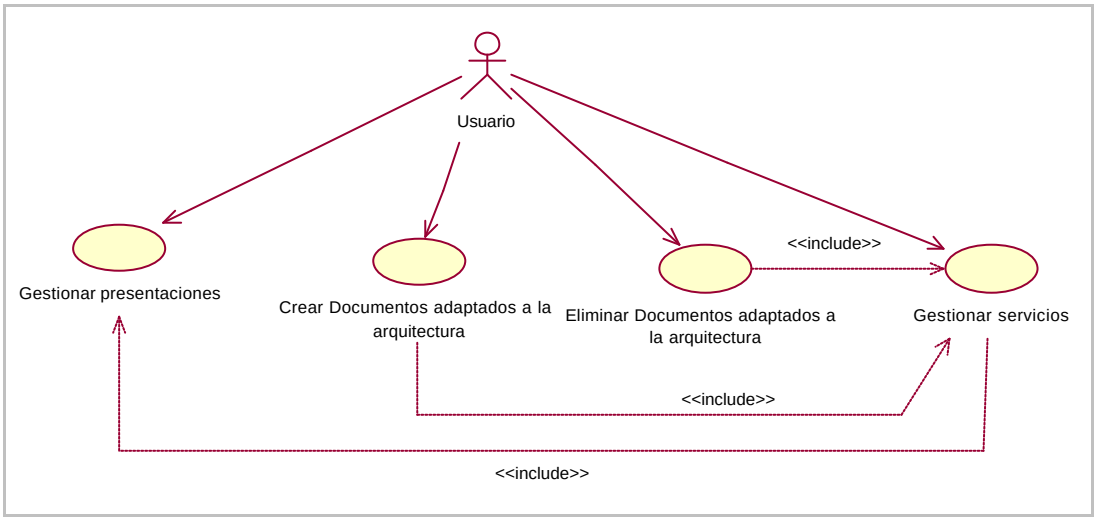


Figura 11: Diagrama de casos de uso de la arquitectura

3.1.3 Diagrama de Clases

El diagrama de clases expresa de manera general la estructura estática del sistema, en términos de clases y sus relaciones. A continuación se muestra el diagrama de clases del sistema que se creó tras la primera aproximación.

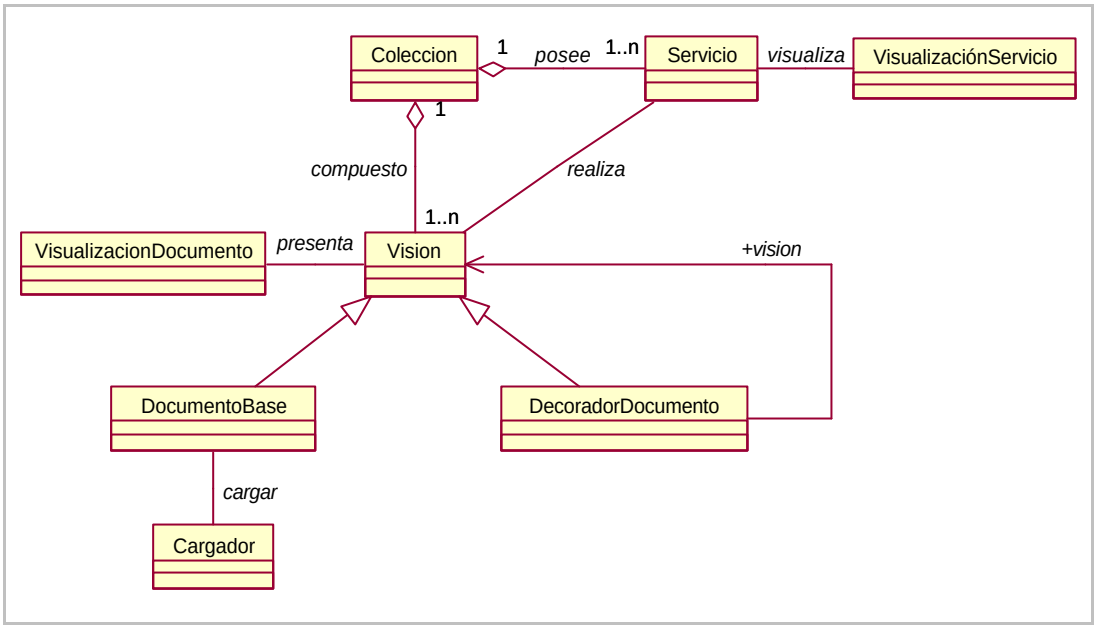
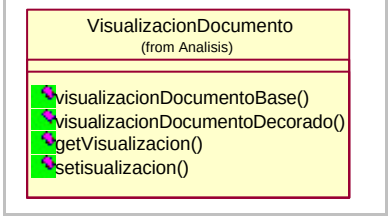
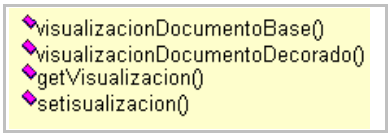
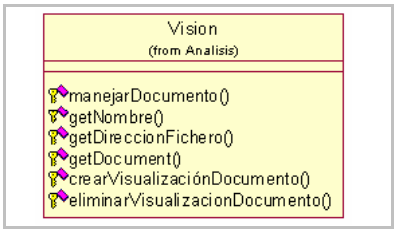
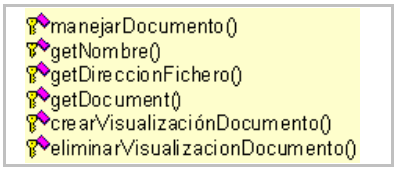
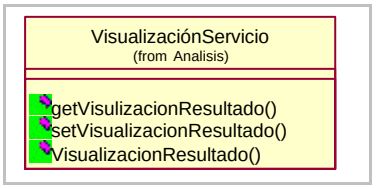


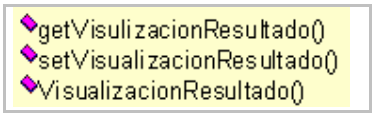
Figura 12: Análisis: Diagrama de clases

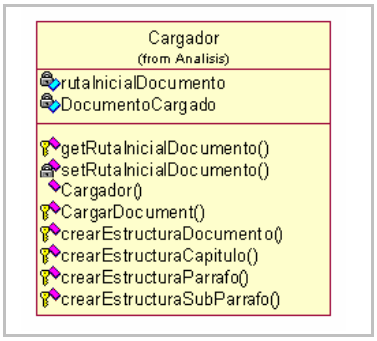
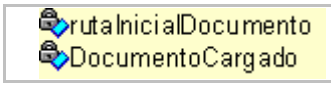
3.1.4 Diccionario de clases

CLASE VISUALIZACION DOCUMENTO	
 **Figura 13:** Análisis: Clase Visualización documento	Clase que se encarga de crear las visualizaciones de los documentos base y decorados.
ATRIBUTOS	
	No posee atributos
MÉTODOS	
 **Figura 14:** Análisis: Métodos Clase VisualizacionDocumento	VisualizacionDocumentoBase: método que permite generar la presentación de los documentos base VisualizacionDocumentoDecorado: método que permite generar la presentación de los documentos decorados getVisualizacion: devuelve la presentación setVisualizacion: asigna la presentación

CLASE VISION	
 **Figura 15:** Análisis: Clase Vision	Clase que representa la interfaz de los documentos gestionados por la arquitectura, tanto los documentos de partida como los derivados de éstos como resultado de aplicar algún tipo de procesamiento sobre ellos. Lo que hace es poseer todos los métodos necesarios para crear los documentos base o bien los documentos decorados que se van a crear en función de un documento base.
ATRIBUTOS	
	Esta clase no posee atributos.
MÉTODOS	
 **Figura 16:** Análisis: Métodos clase Visión	manejarDocumento: permite hacer la llamada para tratar un documento ya bien sea documento base ya bien sea documento decorado. getNombre: devuelve el nombre del documento base getDireccionFichero: devuelve la ruta del documento getDocument: devuelve la información extraída del documento base crearVisualizacionDocumento: permite crear la visualización del documento eliminarVisualizacionDocumento: elimina la visualización de un documento

CLASE VISUALIZACION SERVICIO	
 **Figura 17:** Análisis: Clase VisualizacionServicio	Clase que representa la interfaz del documento XML asociado a un documento

ATRIBUTOS	
	No posee atributos
MÉTODOS	
 **Figura 18:** Análisis: Métodos Clase Visualización Servicios	getVisualizacionResultado: devuelve la presentación generada setVisualizaciónResultado: asigna una presentación VisualizacionResultado: método abstracto que genera la presentación del resultado de la búsqueda de una consulta a través de un servicio

CLASE CARGADOR	
 **Figura 19:** Análisis: Clase Cargador	Clase que se encarga de construir y cargar los documentos de partida, es decir se encarga de extraer la información de un documento original.
ATRIBUTOS	
 **Figura 20:** Análisis: Atributos clase Cargador	rutaInicialDocumento: es la ruta del documento original DocumentoCargado: es el nombre del documento que se va a cargar
MÉTODOS	
	getRutaInicialDocumento: devuelve la ruta del documento original setRutaInicialDocumento: asigna la ruta del documento original al atributo.

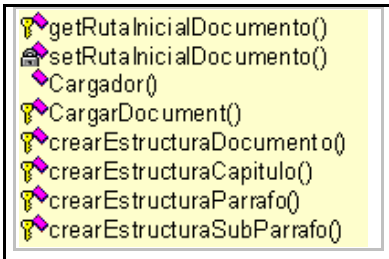


Figura 21: Análisis: Métodos clase Cargador

Cargador: es el constructor de la clase

CargarDocument: este es un método abstracto que deberá de ser implementado en cada una de las subclases. Es el método responsable de hacer la llamada para crear la estructura del documento.

crearEstructuraDocumeno: método abstracto encargado de crear la estructura del documento.

crearEstructuraCapitulos: método abstracto encargado de crear la estructura de cada uno de los capítulos del documento.

crearEstructuraParrafo: método abstracto encargado de crear la estructura de cada uno de los párrafos de los capítulos.

crearEstructuraSubParrafo: método abstracto encargado de crear la estructura de cada uno de los subpárrafos de los párrafos.

CLASE DOCUMENTO BASE

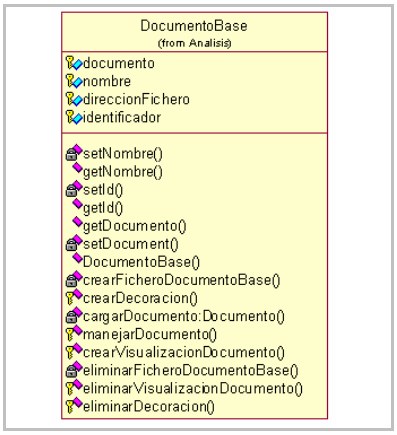


Figura 22: Análisis: Clase Documento Base

Clase que representa a la estructura básica que se obtiene al extraer la información de un documento original y convertirlo a un documento base. Esta clase implementa la interfaz *Vision*.

ATRIBUTOS

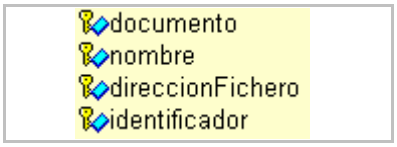


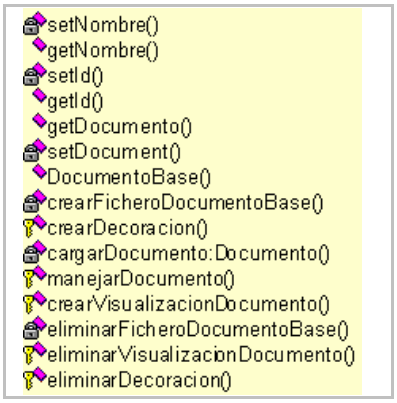
Figura 23: Análisis: Atributos Documento Base

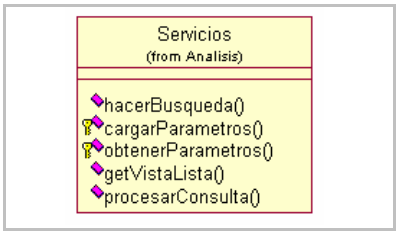
documento: es la información básica que se obtiene al extraer la información de un documento

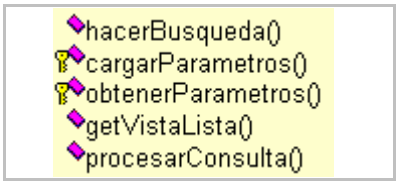
nombre: nombre del fichero que se genera

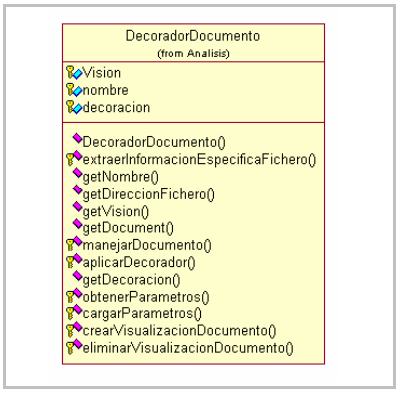
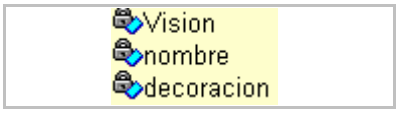
direccionFichero: es la dirección donde se encuentra el fichero

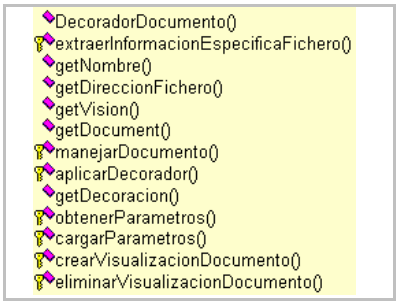
identificador: es el identificador que permite distinguir si dos documentos base son iguales

MÉTODOS	
 **Figura 24:** Análisis: Métodos Documento Base	setNombre: recibe el nombre que se va a asignar al documento base getNombre: devuelve el nombre del documento base setId: recibe el identificador que se va a asignar al documento base getId: devuelve el identificador del documento base getDocument: devuelve la información que se extrae del documento original setDocument: asigna la información extraída del documento original al atributo documento DocumentoBase: es el constructor y recibe la ruta del documento a extraer, la ruta del documento que se va a crear. crearFicheroDocumentoBase: crea el fichero donde se va a guardar almacenado el documento base cargarDocumento: permite a partir de un fichero que contenga el documento base, extraer la información manejarDocumento: recibe un objeto de tipo *Vision*, si el parámetro es nulo es necesario crear el documento base; si es un *DocumentoBase* hay que crear un documento decorado cuya fuente es el Documento base; y si es un documento decorado es necesario crear un documento decorado cuya fuente es otro documento decorado crearVisualizacionDocumento: crea la visualización de un documento base eliminarVisualizacionDocumento: elimina la visualización de un documento base

CLASE SERVICIO	
 **Figura 25:** Análisis: Clase Servicio	Clase que representa los servicios sobre grupos de documentos que va a proporcionar la arquitectura.

ATRIBUTOS	
	No posee atributos.
MÉTODOS	
 **Figura 26:** Análisis: Métodos Clase Servicios	cargarParametros: método abstracto que carga los parámetros de las implementaciones concretas obtenerParametros: método abstracto que permite obtener el nombre de los parámetros que usa la implementación concreta del servicio procesarConsulta: método abstracto que permite procesar la consulta que se realiza sobre el servicio hacerBusqueda: método abstracto que realiza la búsqueda de documentos mediante una consulta a través de un servicio

CLASE DECORADOR DOCUMENTO	
 **Figura 27:** Análisis: Clase DecoradorDocumento	Clase que representa a la estructura de un documento decorado, es decir, un documento que ha sido sometido a algún tipo de procesamiento. Esta clase implementa la interfaz *Vision*.
ATRIBUTOS	
 **Figura 28:** Análisis: Atributos DecoradorDocumento	Vision: este atributo de tipo *Vision* es el responsable de que el sistema sea capaz de saber como hacer para crear decoraciones de decoraciones. nombre: nombre del fichero que se genera decoracion: este atributo es el responsable de la estructura del documento decorado.

MÉTODOS	
 **Figura 29:** Análisis: Métodos DecoradorDocumento	DecoradorDocumento: es el constructor y recibe la ruta del documento base sobre el que se va a realizar la extracción para crear un nuevo objeto de tipo *DecoradorDocumento*. DecoradorDocumento: es otro constructor y recibe la ruta del documento decorado sobre el que se va a realizar la extracción de la información para crear un nuevo objeto de tipo *DecoradorDocumento* extraerInformaciónEspecificaFichero: método necesario para extraer la información de un fichero decorado. getNombre: devuelve el nombre del documento decorado getDocument: devuelve la información que se extrae del documento original llamando al método de la clase *DocumentoBase*. getDireccionFichero: devuelve la ruta del fichero decorado. getVision: devuelve el objeto *Vision*. getDecoracion: devuelve la información que extrae de un documento decorado. manejarDocumento: este método lo que hace es llamar al método *manejarDocumento* de la clase *DocumentoBase* donde se encuentra implementado. aplicarDecorador: este método lo que hace es aplicar el decorador al parámetro que se le pase. Si es un documento base, aplicará el decorador sobre el; si es un documento decorado, aplicará el decorador sobre él cargarParametros: carga los parámetros indicados por el usuario necesarios para aplicar el decorador. crearVisualizacionDocumento: crea la visualización de un documento decorado. eliminarVisualizacionDocumento: elimina la visualización de un documento decorado.

3.1.5 Casos de uso y diagramas de secuencia

Los objetivos de los casos de uso son los siguientes:

- Capturar los requisitos funcionales del sistema y expresarlos desde el punto de vista del usuario.
- Guiar todo el proceso de desarrollo del sistema de información.

Por lo tanto los casos de uso proporcionan un modo claro y preciso de comunicación entre cliente y creador. Desde el punto de vista del cliente, proporciona

una visión de "caja negra" del sistema, esto es, cómo aparece el sistema desde el exterior sin necesidad de entrar en los detalles de su construcción. Para los creadores, esto supone el punto de partida y el eje sobre el que se apoya todo el desarrollo del sistema en su proceso de análisis y diseño.

Un caso de uso es una secuencia de acciones realizadas por el sistema, que producen un resultado observable y valioso para el usuario en particular, es decir, que representa un comportamiento del sistema con el fin de dar respuestas a los usuarios. Aquellos casos de uso que resulten demasiado complejos, se pueden descomponer en un segundo nivel.

El diagrama de secuencia es un tipo de diagrama de interacción cuyo objetivo es describir el comportamiento dinámico del sistema de información haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos.

Un diagrama de secuencia tiene dos dimensiones, el eje vertical representa el tiempo y el eje horizontal, los diferentes objetos. El tiempo avanza desde la parte superior del diagrama hasta la inferior. Normalmente, en relación al tiempo sólo es importante la secuencia de los mensajes, sin embargo, en aplicaciones en tiempo real se podría introducir una escala en el eje vertical.

Respeto a los objetos, es irrelevante el orden en el que se representan, aunque su colocación debería poseer la mayor claridad posible. Cada objeto tiene asociado una línea de vida y focos de control. La línea de vida indica el intervalo de tiempo durante el que existe ese objeto. Un foco de control o activación muestra el período de tiempo en el cual el objeto se encuentra ejecutando alguna operación, ya sea directamente o mediante un procedimiento concurrente.

Un escenario es una secuencia de acciones que ilustra un comportamiento. Un escenario puede utilizarse para ilustrar la interacción o la ejecución de una instancia de un caso de uso.

A continuación se presentan los casos de uso y sus correspondencias con los escenarios principales identificados en el sistema.

3.1.5.1 Caso de uso: Crear Documentos adaptados a la arquitectura

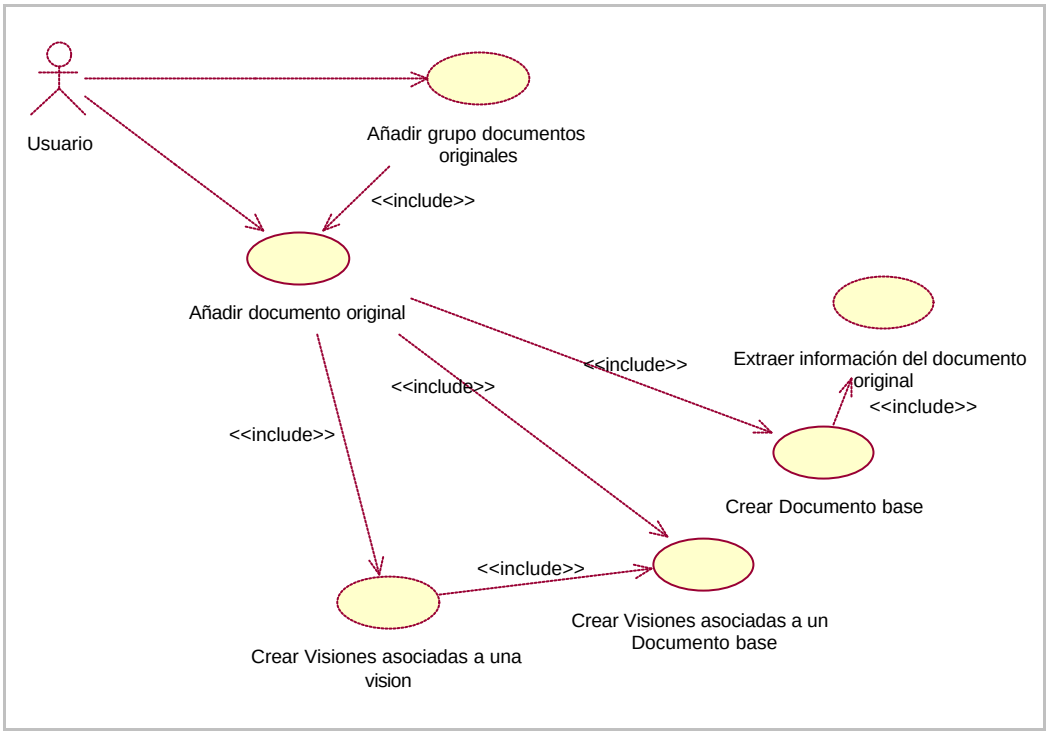


Figura 30: Análisis: Caso de uso: Crear Documentos adaptados a la arquitectura

OBJETIVOS
Este caso de uso tiene la funcionalidad de que a partir de un grupo de documentos que le pasa el usuario, la arquitectura extrae la información de éstos para luego generar los documentos base asociados. A partir de estos últimos es capaz de crear las visiones asociadas y las visiones de las propias visiones (siempre siguiendo la indicación del usuario).
ACTORES
Usuario
PRECONDICIONES
Se requiere que el usuario indique todos os parámetros necesarios para la realización de este caso de uso.
POSTCONDICIONES
Se generan los documentos base asociados a los documentos originales pasados así como las visiones asociadas, y las visiones asociadas a esas visiones (siempre dependiendo de la especificación del usuario).

FUNCIONES QUE CUMPLE
1, 2, 5, 6, 7

Escenario: Añadir un grupo de documentos originales

Este escenario no tiene casi mayor relevancia, ya que la única funcionalidad que posee es justamente invocar, por cada uno de los documentos pasados por el usuario, al escenario "*Añadir un documento original*", que se muestra a continuación.

Escenario: Añadir un documento original

La única funcionalidad que posee este escenario es invocar la arquitectura pasándole como parámetro la dirección donde se encuentra el documento que se desea adaptar a la arquitectura. Todas las tareas que se van a realizar una vez indicada su ubicación se explican a continuación.

Escenario: Crear documento Base

DESCRIPCIÓN
Este escenario ocurre cuando el usuario le indica al sistema información sobre la dirección donde se encuentra el documento que se quiere incorporar al sistema, creando un documento base que lo represente, o bien, cuando el usuario indica el directorio donde se encuentran los documentos originales y se realiza este escenario para cada uno de ellos.
FLUJO PRINCIPAL
1. El sistema comprueba que exista el documento original. 2. El sistema comprueba que no exista un documento base ya asociado al documento original que le está indicando el usuario. 3. El sistema carga el documento original que le pasa el usuario y pasa a extraer toda la información posible que éste posee.(ver siguiente escenario) 4. Con la información anteriormente extraída, se genera un nuevo documento gestionado internamente por la arquitectura que se llamará documento base
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento original no existe]: el documento original que le fue pasado por el usuario no existe. - [Excepción: Documento ya existente]: el documento que va a ser

introducido ya existe en el sistema.

- [Excepción: No se pudo cargar el documento original]: la arquitectura indica que le fue imposible extraer la información del documento original.
- [Excepción: No se pudo crear el documento base]: se produjo un error en el momento de crear el documento base que iba a poseer la información que anteriormente se había extraído.

PRECONDICIONES

La única precondición es que el usuario le tiene que haber indicado cual es el documento original a partir del cual se van a realizar todas estas tareas o bien indicar el directorio donde se encuentran los documentos originales sobre los que se quiere crear los documentos base asociados.

POSTCONDICIONES

El/los documento/s base fue/fueron creado/s.

Es necesario indicar que este escenario va a ser el responsable de realizar la llamada al escenario *"Añadir un documento a un servicio sobre grupo de documentos"* cuando la información que gestione dicho servicio esté basada en documentos base. Se detalla a continuación en el apartado "*Gestionar Servicios*".

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

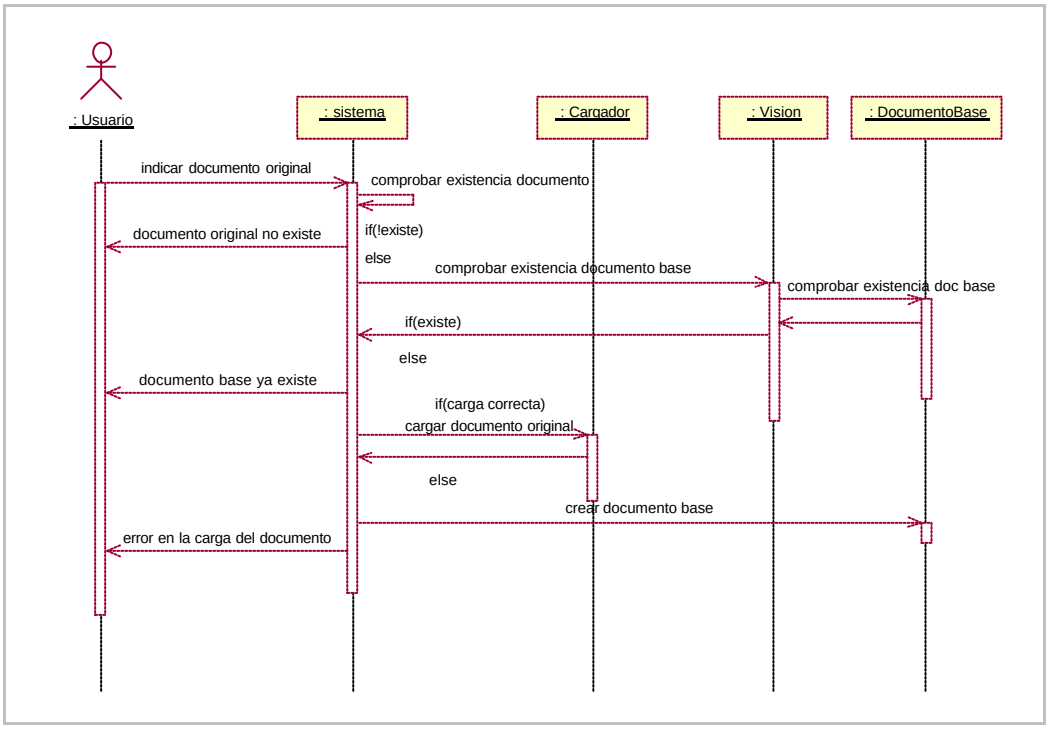


Figura 31: Análisis: Diagrama de secuencia: Crear Documento Base

El diagrama de colaboración de este escenario:

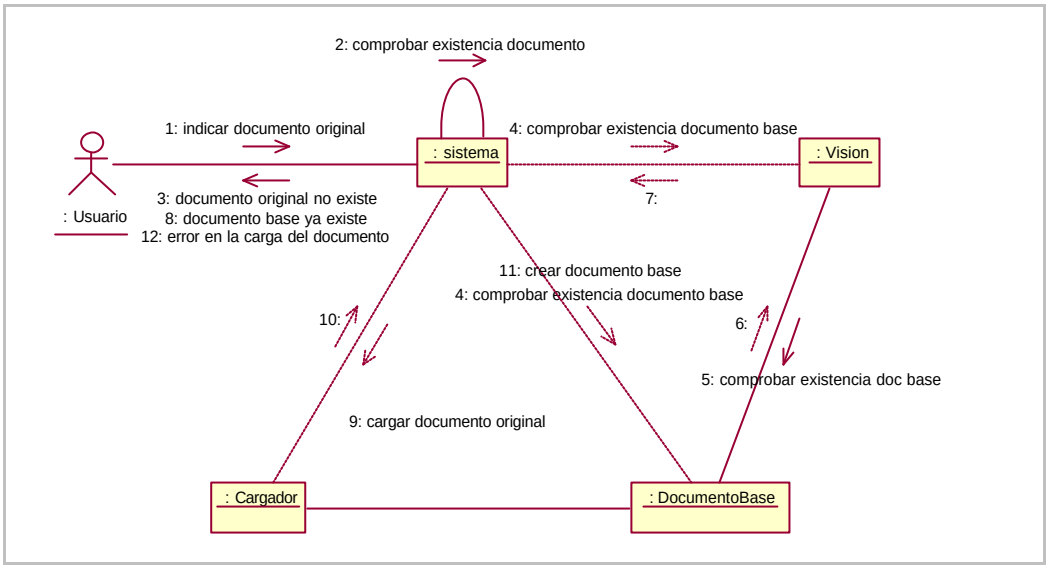


Figura 32: Análisis: Diagrama de colaboración: Crear Documento Base

Escenario: Extraer información del documento original

DESCRIPCIÓN
Este escenario ocurre cuando la arquitectura pretende crear un documento base, es decir, después de que el usuario le indique donde se encuentra el documento que se quiere incorporar al sistema.
FLUJO PRINCIPAL
1. El sistema comprueba el tipo de documento que le indica el usuario. 2. Según el tipo de documento, es decir, de la extensión del documento, pasa a extraer el título del documento. 3. Extrae los capítulos del documento. 4. Para cada capitulo, extrae el subtitulo si lo posee y en todo caso extrae los párrafos que lo componen.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento original no existe]: el documento original que le fue pasado por el usuario no existe. - [Excepción: No se pudo cargar el documento original]: la arquitectura indica que le fue imposible extraer la información del documento original.
PRECONDICIONES

No existen precondiciones
POSTCONDICIONES
Devuelve la estructura generada tras la extracción de la información.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

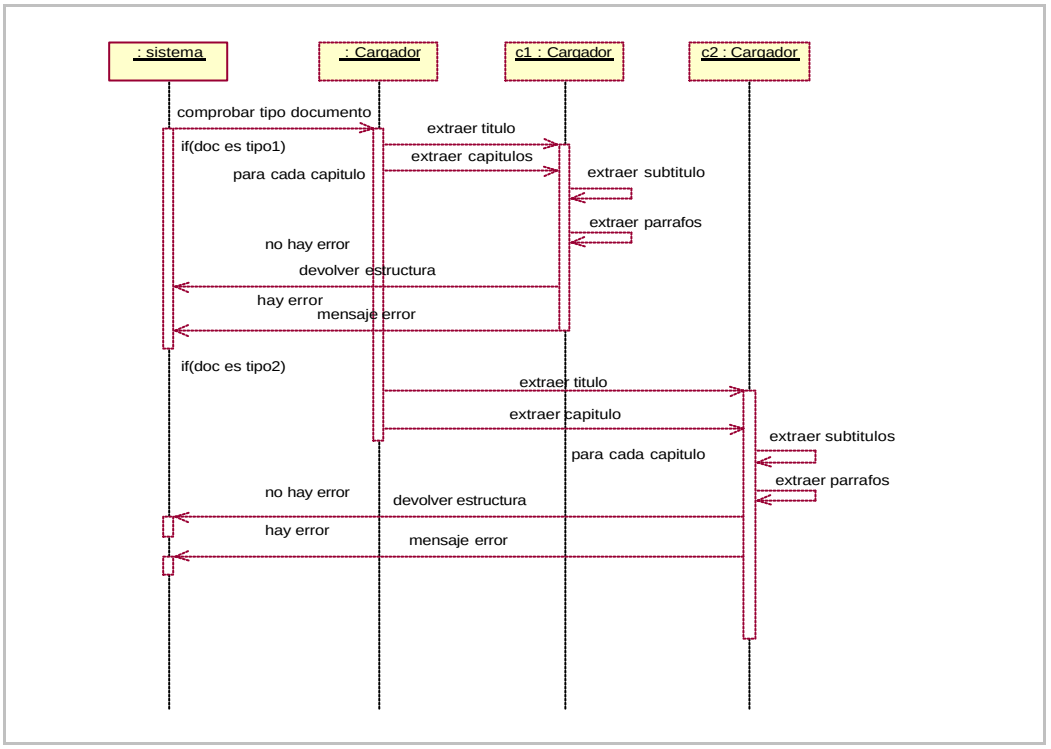


Figura 33: Análisis: Diagrama de secuencia: Extraer información del documento original

El diagrama de colaboración de este escenario:

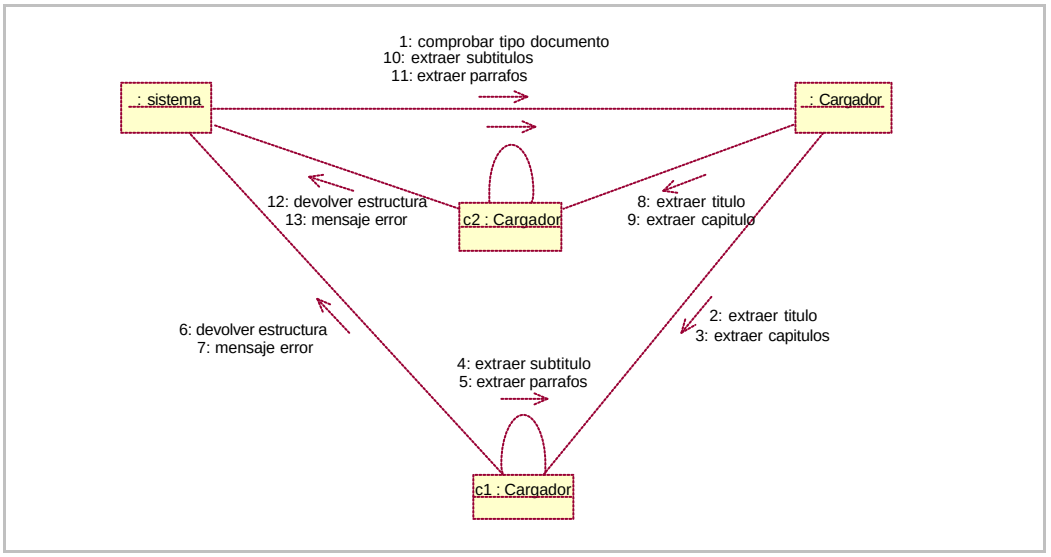


Figura 34: Análisis: Diagrama de colaboración: Extraer información del documento original

Escenario: Crear visiones asociadas a un documento base

DESCRIPCIÓN
Este escenario ocurre cuando la arquitectura ha generado el documento base y el usuario indicó que las fuentes de los servicios de documentos individuales proceden de un documento base, creando de este modo las visiones asociadas al documento base.
FLUJO PRINCIPAL
Para cada uno de los servicios sobre documentos individuales que se encuentre en el fichero de configuración cuya fuente sea un documento base: 1. El sistema comprueba exista el documento base a partir del cual se quieren generar las visiones. 2. A partir del documento base, extrae su contenido. 3. Generar la visión a partir del contenido del documento base, es decir, se genera el documento decorado
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento base no existe]: el documento base pasado no existe. - [Excepción: No se pudo crear el documento decorado]: se produjo un error en el momento de crear el documento decorado.
PRECONDICIONES
La única precondición que se debe de cumplir es que se tiene que haber creado previamente el documento base sobre el que se desea realizar las decoraciones, es decir, sobre el que se van a generar las visiones dependiendo de la especificación del usuario.
POSTCONDICIONES
Se ha creado tantas visiones como servicios sobre documentos individuales, cuya fuente sea el documento base, se hayan especificado. Es necesario indicar que este escenario va a ser el responsable de realizar la llamada al escenario *"Añadir un documento a un servicio sobre grupo de documentos*" cuando la información que utilice dicho servicio se basa en documentos decorados de esa visión. Se detalla a continuación en el apartado "*Gestionar Servicios*".

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

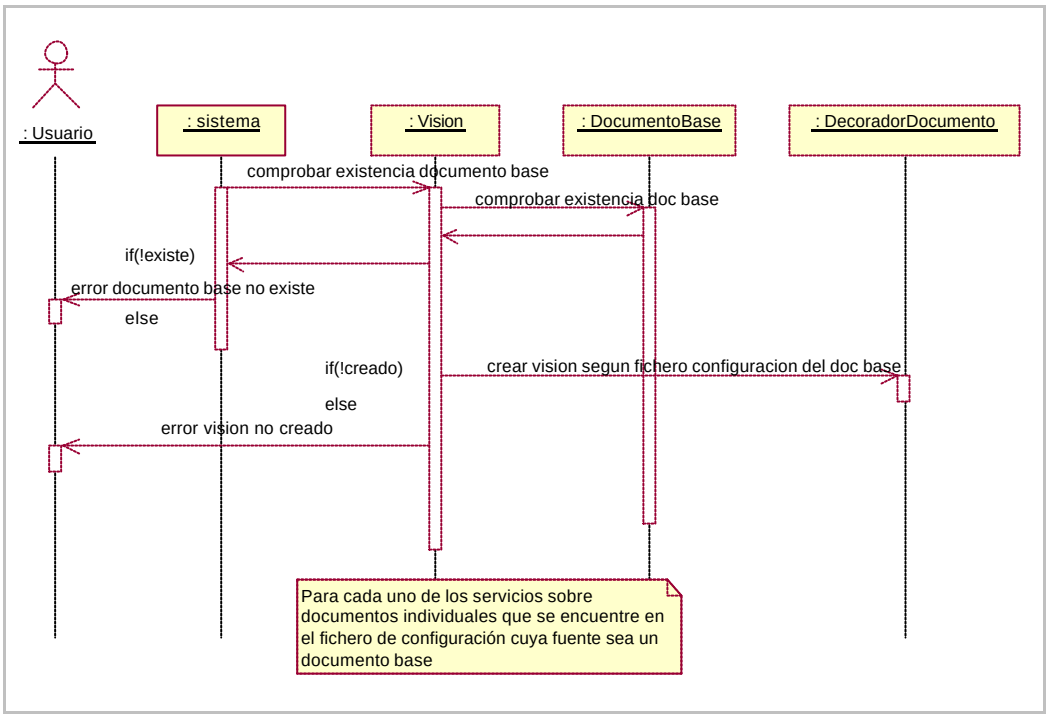


Figura 35: Análisis: Diagrama de secuencia: Crear visiones asociadas a un documento base

El diagrama de colaboración de este escenario:

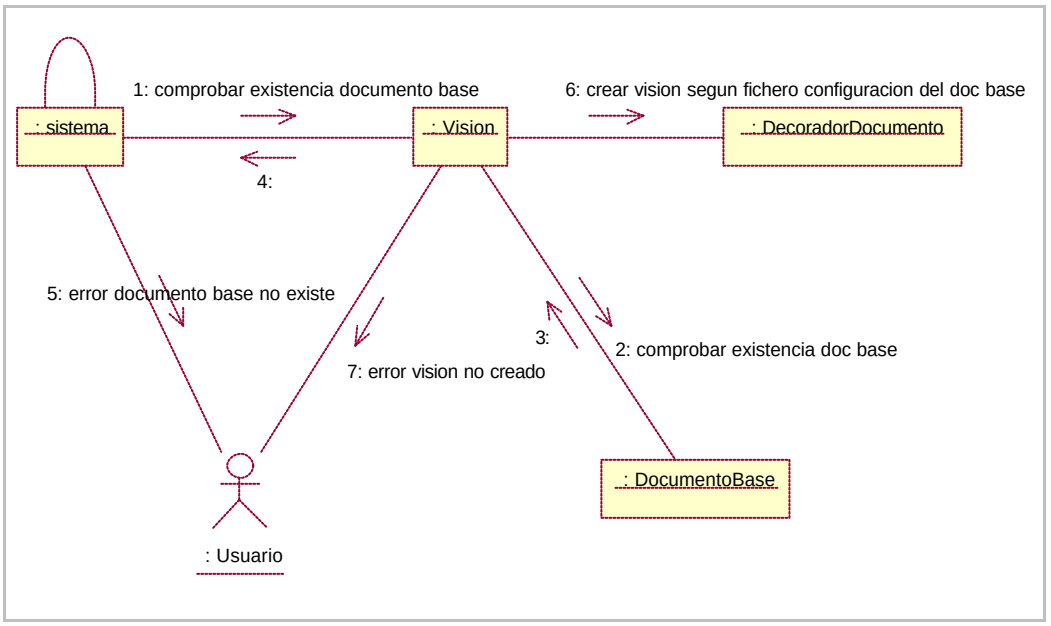


Figura 36: Análisis: Diagrama de colaboración: Crear visiones asociadas a un documento base

Escenario: Crear visiones asociadas a una visión

DESCRIPCIÓN
Este escenario ocurre cuando la arquitectura ha generado el documento base, las visiones asociadas al documento base ya está creadas y el usuario haya especificado que las fuentes de los servicios de documentos individuales proceden de otra visión, creando de este modo las visiones asociadas a esas visiones.
FLUJO PRINCIPAL
Para cada uno de los servicios sobre documentos individuales que se encuentre en el fichero de configuración cuya fuente sea una visión: 1. El sistema comprueba si existe la visión que se encuentra como fuente en el servicio sobre documentos individuales. 2. Si existe: A partir de la visión que se encuentra como fuente en servicio sobre documentos individuales que se encuentra en el fichero de configuración, se genera la visión asociada. 3. Si no existe: Se crea de un modo recursivo la visión que se encuentra como fuente en servicio sobre documentos individuales que se encuentra en el fichero de configuración hasta alcanzar como fuente el documento base (ver apartado anterior).
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No se pudo crear el documento decorado]: se produjo un error en el momento de crear el documento decorado.
PRECONDICIONES
La única precondición que se debe de cumplir es que se tiene que haber creado previamente el documento base sobre el que se desea realizar la decoración, es decir, sobre el que se van a generar las visiones dependiendo de la especificación realizada por parte del usuario.
POSTCONDICIONES
Se ha creado tantas visiones como servicios sobre documentos individuales, cuya fuente sea una visión, hayan sido especificados. Es necesario indicar que este escenario va a ser el responsable de realizar la llamada al escenario *"Añadir un documento a un servicio sobre grupo de documentos*" cuando la información que utiliza dicho servicio se basa en documentos decorados de esa visión. Se detalla a continuación en el apartado "*Gestionar Servicios*".

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

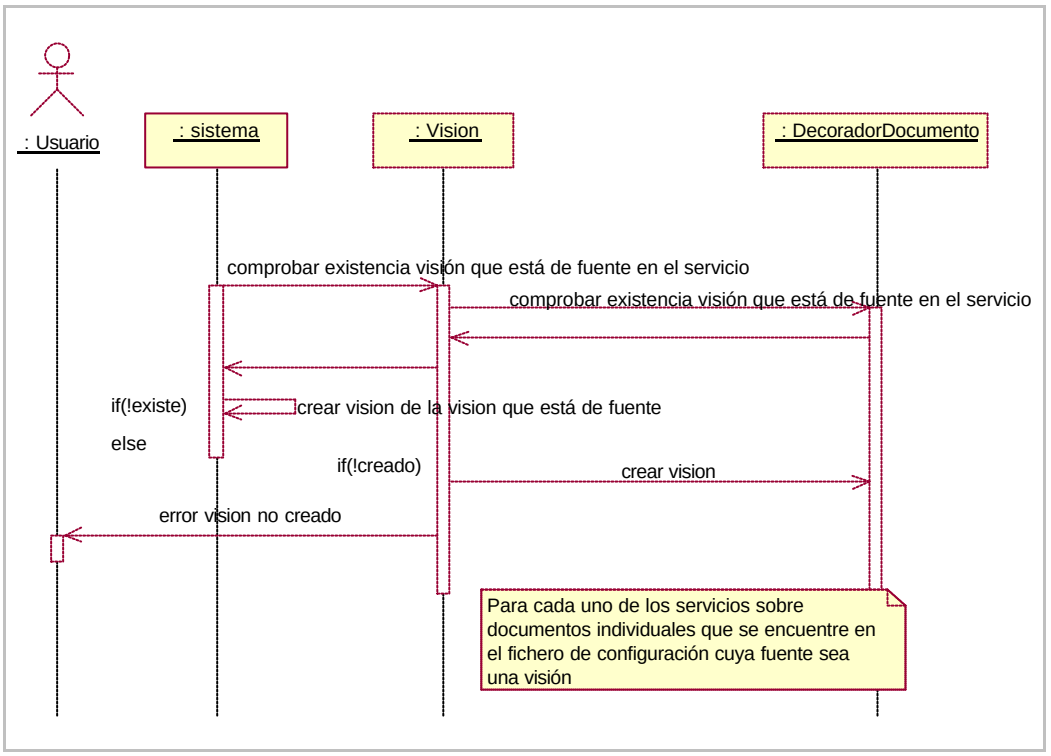


Figura 37: Análisis: Diagrama de secuencia: Crear visiones asociadas a una visión

El diagrama de colaboración de este escenario:

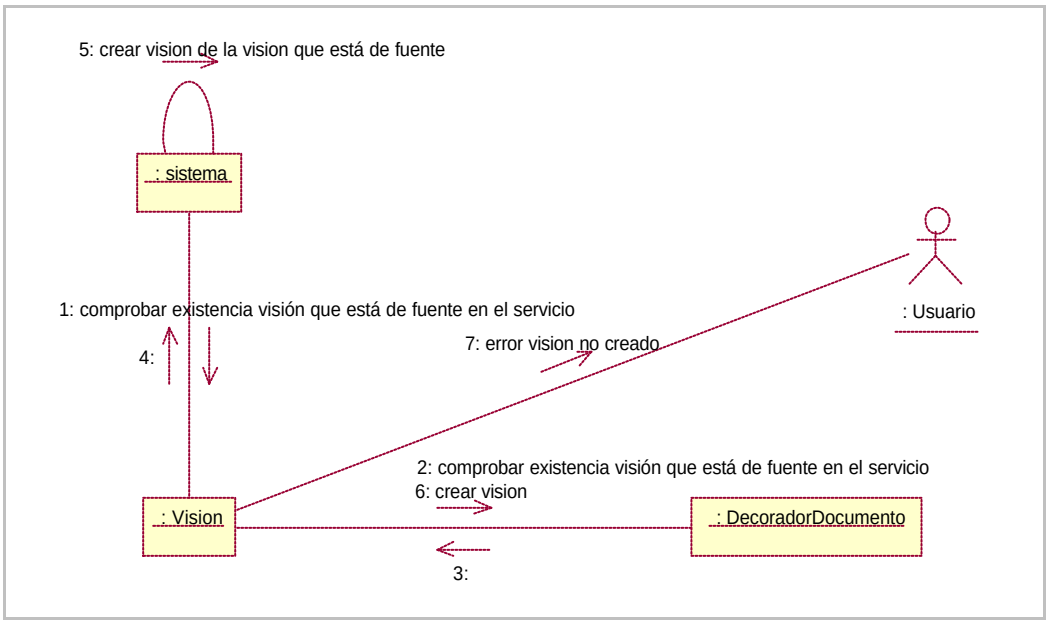


Figura 38: Análisis: Diagrama de colaboración: Crear visiones asociadas a una visión

3.1.5.2 Caso de uso: Eliminar Documento adaptado a la arquitectura

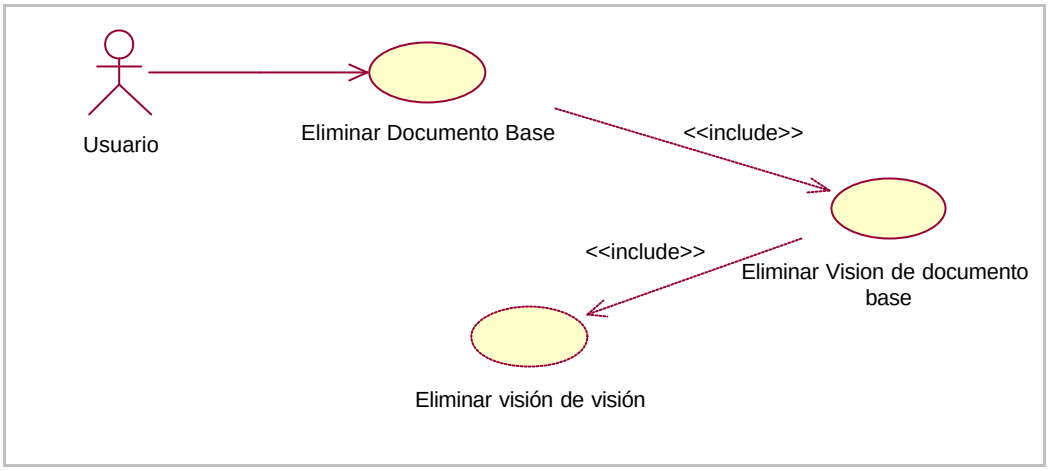


Figura 39: Análisis: Caso de uso: Eliminar Documento adaptado a la arquitectura

OBJETIVOS
Este caso de uso permite eliminar el documento base que le es indicado por el usuario, pero además también elimina todas las visiones que existen asociadas a ese documento base.
ACTORES
Usuario
PRECONDICIONES
Se requiere que el usuario haya indicado los parámetros necesarios para la realización de este caso de uso. Es necesario haber creado previamente los documentos base así como sus visiones para poder ser eliminadas
POSTCONDICIONES
El resultado es la eliminación del documento base indicado por el usuario así como todas aquellas visiones que se han generado teniendo como base dicho documento base indicado por el usuario (siempre dependiendo de las especificaciones del usuario).
FUNCIONES QUE CUMPLE
14, 15, 16

Escenario: Eliminar Documento Base

DESCRIPCIÓN
Este escenario ocurre cuando el usuario le indica a la arquitectura que desea eliminar uno de los documentos base que existen en la colección.
FLUJO PRINCIPAL
1. El sistema comprueba si existe el documento base indicado por el usuario. 2. El sistema procede a eliminar las visiones asociadas al documento base (ver el apartado siguiente). 3. El sistema procede a eliminar el documento base que le indicó el usuario.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No se pudo eliminar el documento base]: se produjo un error en el momento de eliminar el documento base.
PRECONDICIONES
No existen precondiciones
POSTCONDICIONES
Se han eliminado los documentos decorados, cuya fuente fuese el documento base, pero además se ha eliminado todos los documentos decorados que tenían por fuente a las visiones anteriores, además de eliminar el propio documento base. Es necesario indicar que este escenario va a ser el responsable de realizar la llamada al escenario *"Eliminar un documento de un servicio sobre grupo de documentos"* cuando la información que almacene dicho servicio sea sobre documentos base. Se detalla a continuación en el apartado "*Gestionar Servicios*".

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

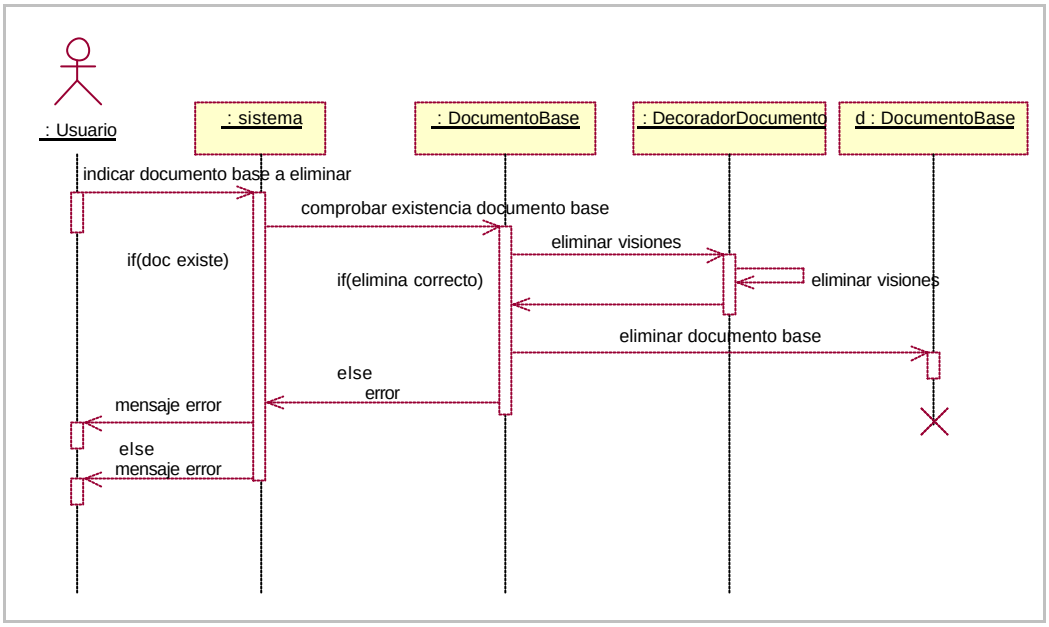


Figura 40: Análisis: Diagrama de secuencia: Eliminar Documento Base

El diagrama de colaboración de este escenario:

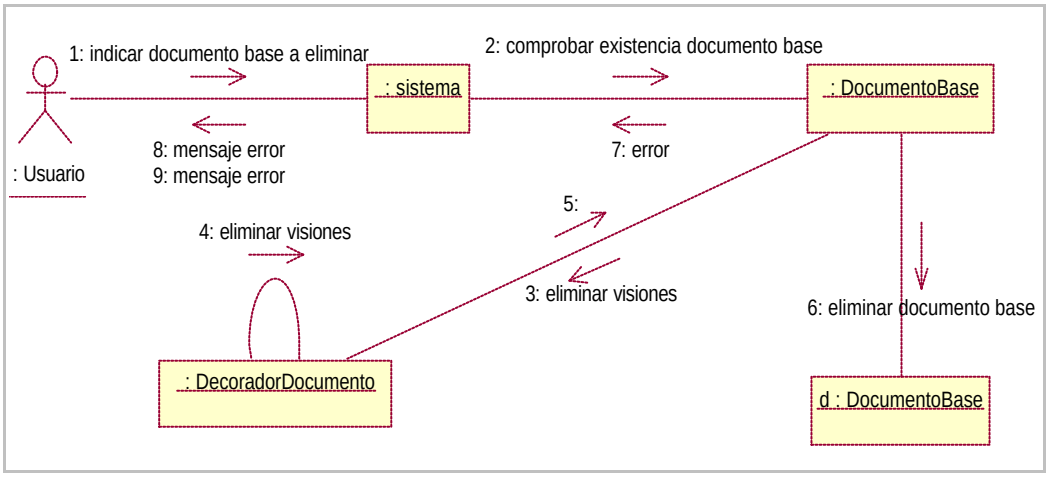


Figura 41: Análisis: Diagrama de colaboración: Eliminar Documento Base

Escenario: Eliminar visión de documento base

DESCRIPCIÓN
Este escenario ocurre cuando se procede a eliminar un documento base, eliminándose las visiones que tienen por fuente dicho documento base.
FLUJO PRINCIPAL
1. El sistema comprueba si existe visiones cuya fuente sea el documento base indicado por el usuario. 2. Si existen: Para cada uno de los documentos decorados que tienen por

fuente el documento base, el sistema procede a eliminar las visiones que posean como fuente esa visión (ver apartado siguiente) y elimina cada uno de los documentos decorados que tienen por fuente al documento base.

3. Si no existen: no hace nada.

FLUJO EXCEPCIONAL DE EVENTOS

Se distinguen los siguientes:

- [Excepción: No se pudo eliminar el documento decorado]: se produjo un error en el momento de eliminar el documento decorado

PRECONDICIONES

No existe precondición

POSTCONDICIONES

Se ha eliminado todas las decoraciones de documentos que poseían como fuente a la visión que se le pasa inicialmente además de eliminar la visión pasada.

Es necesario indicar que este escenario va a ser el responsable de realizar la llamada al escenario "*Eliminar un documento de un servicio sobre grupo de documentos*" cuando la información que utilice dicho servicio tenga como base los documentos decorados de esa visión. Se detalla a continuación en el apartado "*Gestionar Servicios*".

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

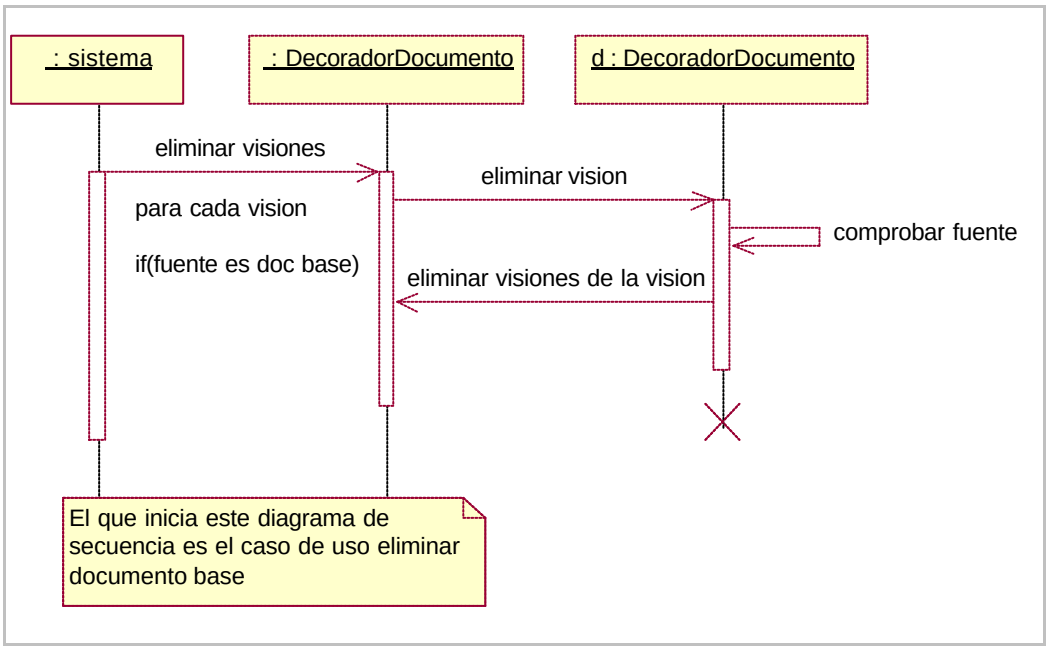


Figura 42: Análisis: Diagrama de secuencia: Eliminar visión de documento base

El diagrama de colaboración de este escenario:

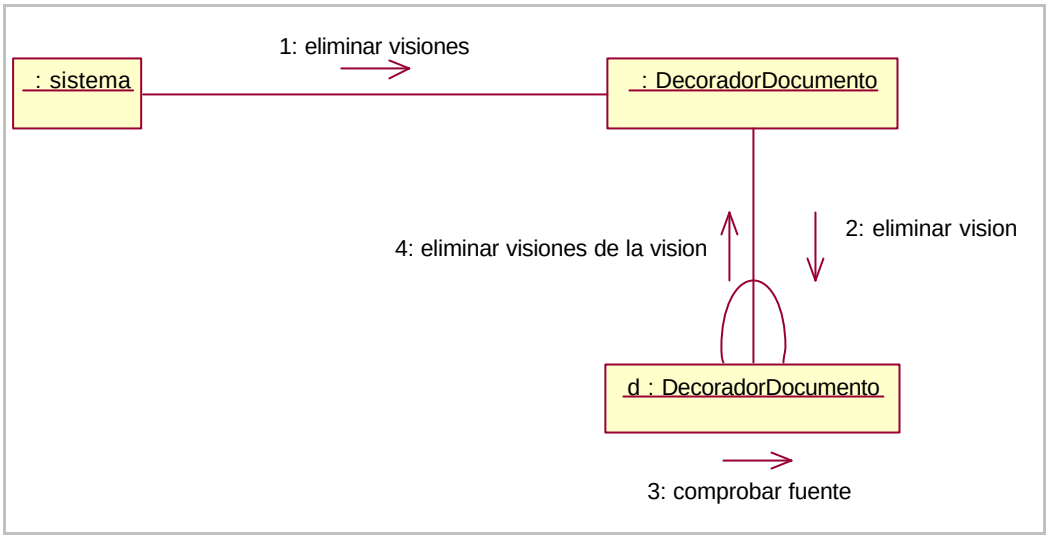


Figura 43: Análisis: Diagrama de colaboración: Eliminar visión de documento base

Escenario: Eliminar visión de visión

DESCRIPCIÓN
Este escenario ocurre cuando se procede a eliminar una visión, eliminándose también las visiones que tienen por fuente dicha visión.
FLUJO PRINCIPAL
1. El sistema comprueba si existe visiones cuya fuente sea la visión que se va a eliminar. 2. Si existen: El sistema procede a eliminar cada uno de los documentos decorados que tienen por fuente a la visión que le está indicando el sistema. 3. Si no existen: no hace nada.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No se pudo eliminar el documento decorado]: se produjo un error en el momento de eliminar el documento decorado
PRECONDICIONES
No existe precondición
POSTCONDICIONES
Se ha eliminado los documentos decorados cuya fuente es la visión que se le indica.

Es necesario indicar que este escenario va a ser el responsable de realizar la llamada al escenario "*Eliminar un documento de un servicio sobre grupo de documentos*" cuando la información que utilice dicho servicio esté basado en sobre documentos decorados de esa visión. Se detalla a continuación en el apartado "*Gestionar Servicios*".

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

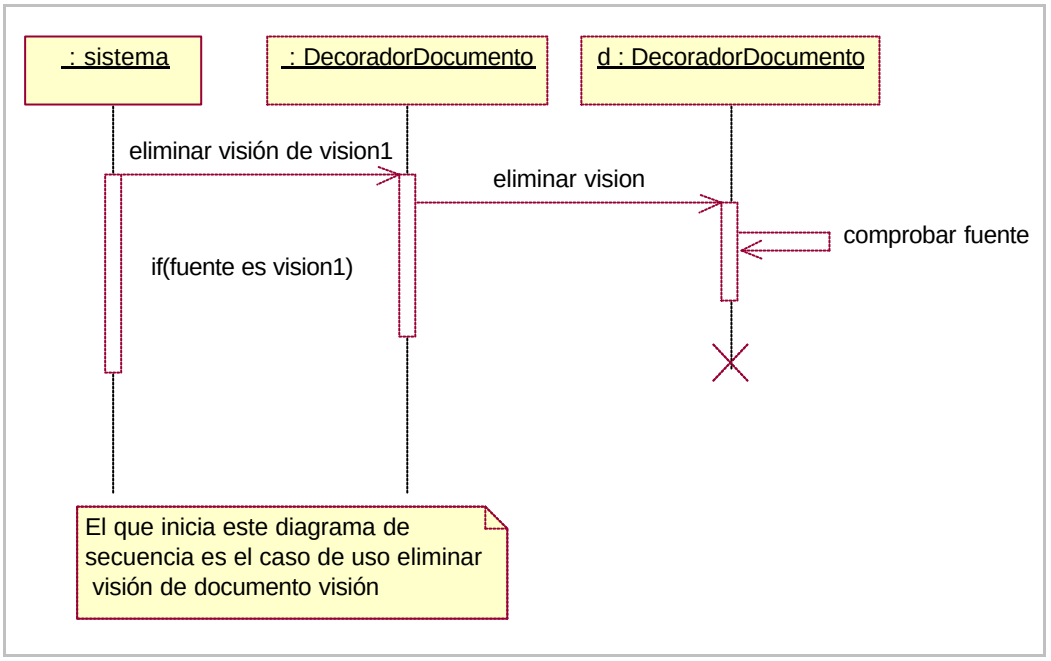


Figura 44: Análisis: Diagrama de secuencia: Eliminar visión de visión

El diagrama de colaboración de este escenario:

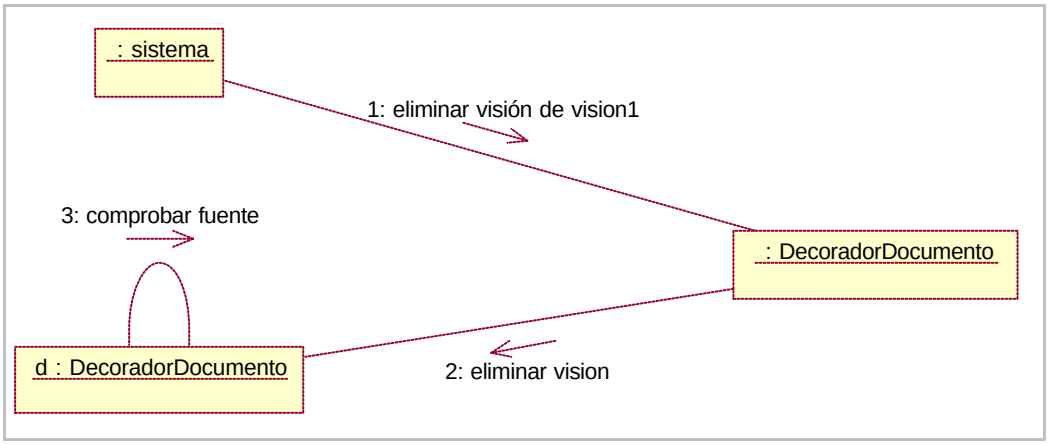


Figura 45: Análisis: Diagrama de colaboración: Eliminar visión de visión

3.1.5.3 Caso de uso: Gestionar Servicios

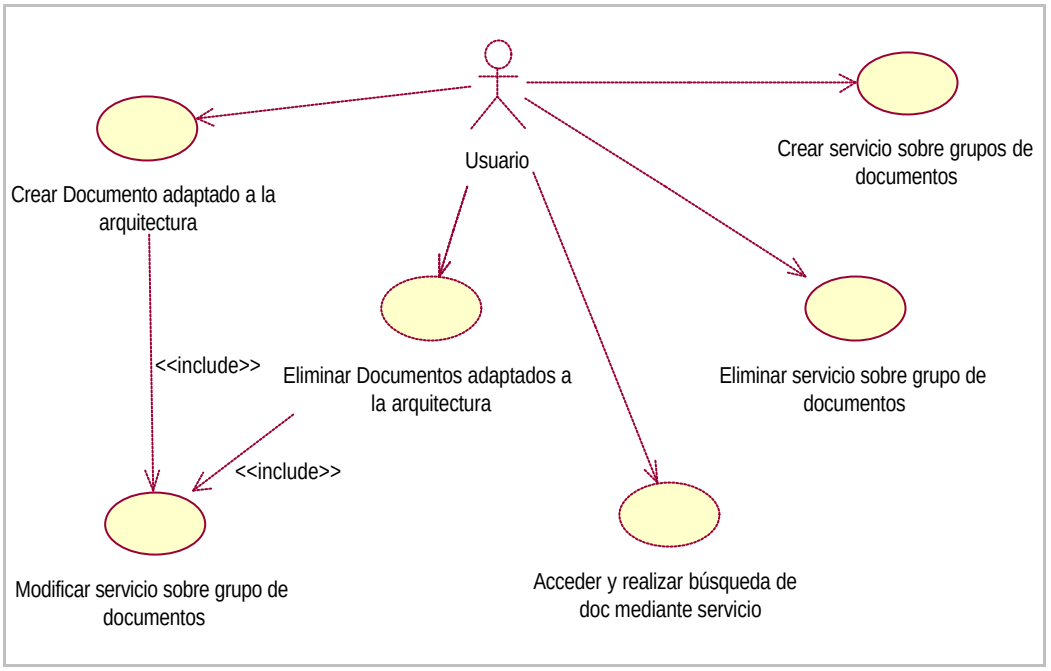


Figura 46: Análisis: Caso de uso: Gestionar Servicios

OBJETIVOS
Este caso de uso gestiona todos los servicios que indique inicialmente el usuario. Es decir, permite crearlos, eliminarlos y modificarlos. Además permite acceder a ellos y realizar el acceso a los documentos a través de un parámetro o consulta.
ACTORES
Usuario. Los casos de uso "*Crear Documento adaptado a la arquitectura*" y "*Eliminar documentos adaptados a la arquitectura*", se podrían considerar también actores en este caso ya que son los que instancian el caso de uso "*Modificar servicio sobre grupo de documentos*".
PRECONDICIONES
Los parámetros que suministra el usuario deben de ser correctos.
POSTCONDICIONES
No existen.
FUNCIONES QUE CUMPLE
3, 4, 8, 17, 20

Escenario: Crear Servicio sobre grupo de documentos

DESCRIPCIÓN
Este caso de uso permite crear las estructuras necesarias para dar soporte a un servicio sobre grupo de documentos que indique el usuario.
FLUJO PRINCIPAL
1. El usuario proporciona los parámetros necesarios para la creación del servicio 2. El sistema comprueba que no exista dicho servicio. 3. Si no existe, pasa a generar las estructuras necesarias.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No se pudo crear el servicio]: se produjo un error en el momento de la creación del servicio - [Excepción: Servicio ya existente]: el sistema detectó que ya existía un servicio como el que se quería crear.
PRECONDICIONES
Los parámetros que le pasa el usuario deben de ser correctos.
POSTCONDICIONES
Se crea la estructura adecuada del servicio que indica el usuario.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

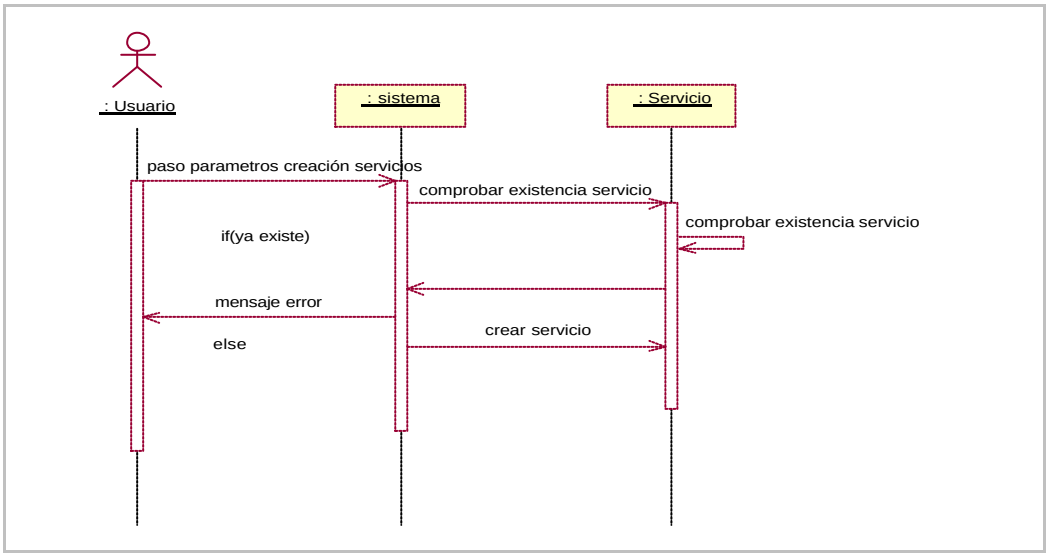


Figura 47: Análisis: Diagrama de secuencia: Crear servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

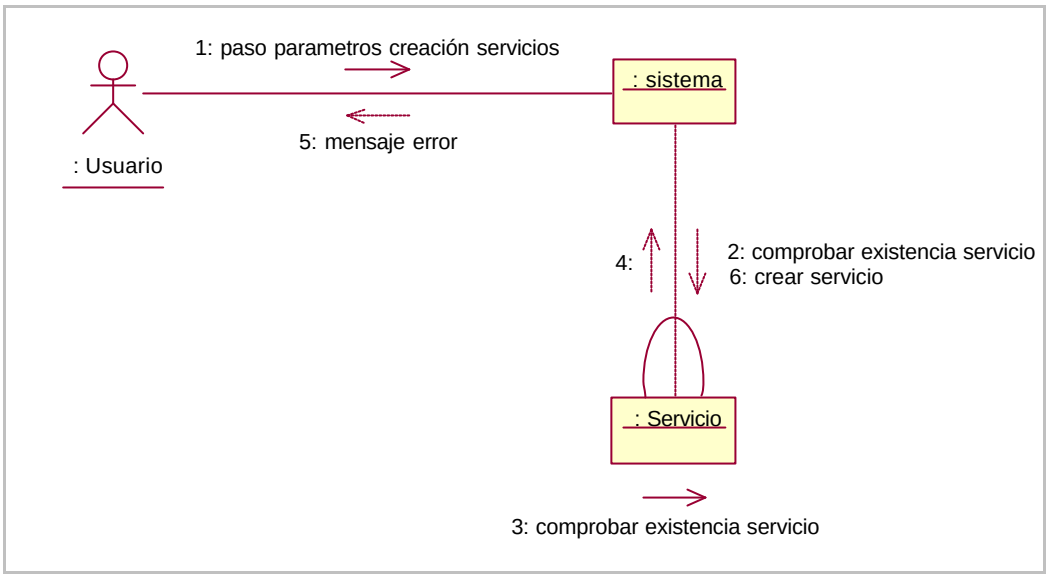


Figura 48: Análisis: Diagrama de colaboración: Crear servicio sobre grupo de documentos

Escenario: Eliminar Servicio sobre grupo de documentos

DESCRIPCIÓN
Este caso de uso permite eliminar las estructuras necesarias utilizadas por un servicio sobre grupo de documentos que indique el usuario.
FLUJO PRINCIPAL
1. El usuario indica al sistema cual es el nombre del servicio que desea eliminar. 2. El sistema comprueba que exista. 3. Si existe, pasa a eliminar las estructuras asociadas a ese servicio sobre grupos de documentos. 4. Si no existe, devuelve un mensaje de error
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El servicio no existe]: el sistema no encontró el servicio que el usuario pedía que se eliminase. - [Excepción: No se pudo eliminar el servicio]: se produjo un error en el momento de eliminar el servicio.

PRECONDICIONES
Los parámetros que le pasa el usuario deben de ser correctos. El servicio debe de estar creado
POSTCONDICIONES
Se eliminan las estructuras creadas por el servicio indicado por el usuario.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

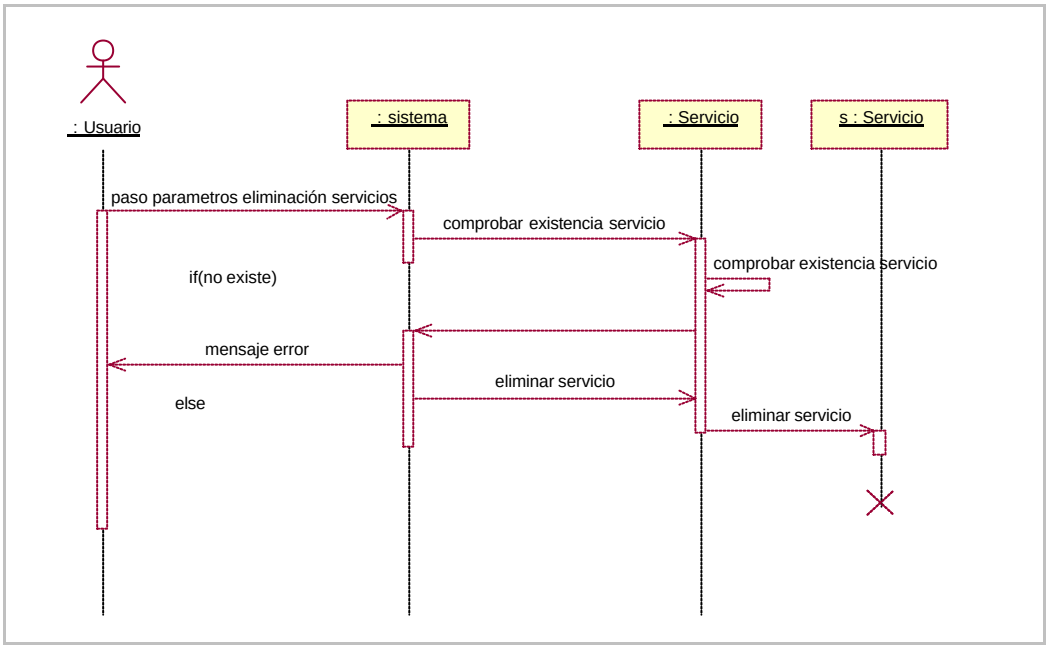


Figura 49: Análisis: Diagrama de secuencia: Eliminar servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

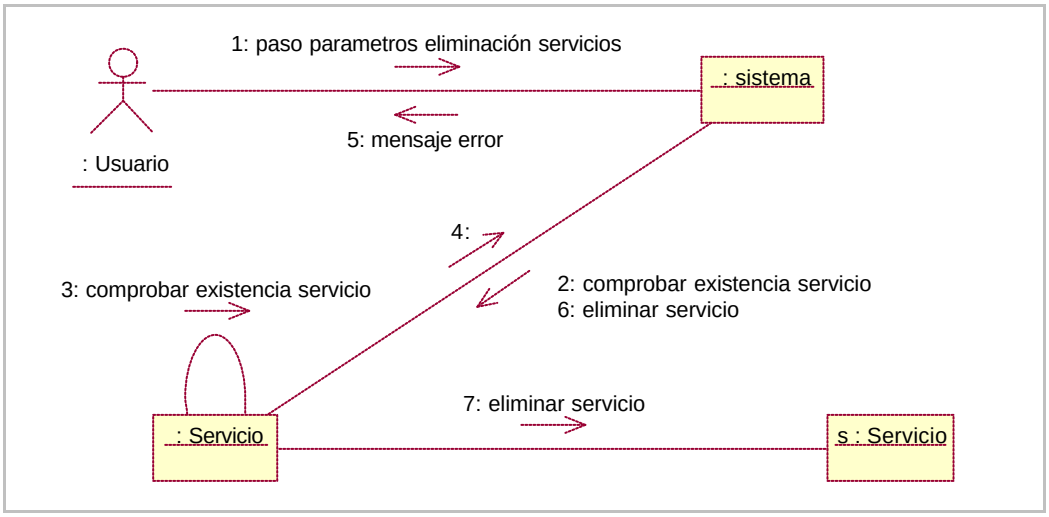


Figura 50: Análisis: Diagrama de colaboración: Eliminar servicio sobre grupo de documentos

Escenario: Modificar Servicio sobre grupo de documentos

Este escenario se puede descomponer en dos. Por un lado "*Añadir un documento a un servicio sobre grupo de documentos*" y por el otro, "E*liminar un documento de un servicio sobre grupos de documentos*". Se muestran ambos a continuación de una forma más detallada.

Escenario: Añadir un documento a un servicio sobre grupo de documentos

DESCRIPCIÓN
Este caso de uso permite modificar las estructuras asociadas a un servicio ya que se pretende añadir la información necesaria para incorporar un nuevo documento (tanto documento base como una visión) al servicio.
FLUJO PRINCIPAL
1. El sistema carga el documento cuya información quiere que se agregue al servicio en cuestión. 2. El sistema extrae la información del documento. 3. El sistema procede a agregar la información pertinente al servicio.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No se pudo agregar el documento al servicio]: se produjo un error en el momento de la modificación de las estructuras del servicio.
PRECONDICIONES
El caso de uso "*Crear documentos adaptados a la arquitectura*" indica cual va a ser el servicio sobre el que se va a realizar una modificación de sus estructuras. El servicio tiene que estar creado. El documento cuya información se quiere incorporar al servicio debe de existir.
POSTCONDICIONES
Se modifica la estructura del servicio agregando la información pertinente del documento.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

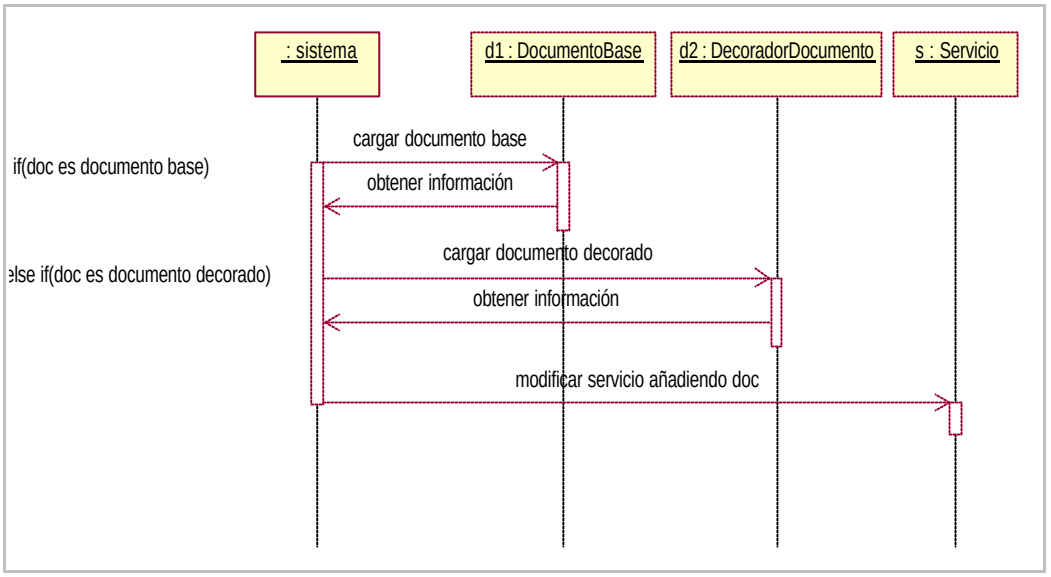


Figura 51: Análisis: Diagrama de secuencia: Añadir un documento a un servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

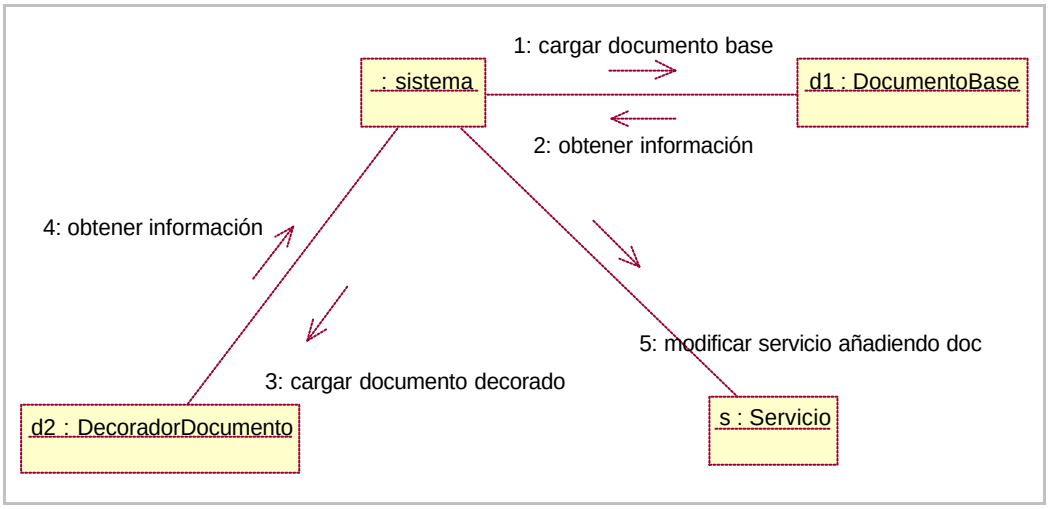


Figura 52: Análisis: Diagrama de colaboración: Añadir un documento a un servicio sobre grupo de documentos

Escenario: Eliminar un documento de un servicio sobre grupo de documentos

DESCRIPCIÓN
Este caso de uso permite modificar las estructuras asociadas a un servicio, ya que se pretende eliminar la información asociada a un documento (tanto documento base como una visión) gestionado por el servicio.

FLUJO PRINCIPAL
1. El sistema carga el servicio indicado por el caso de uso "*Eliminar documento adaptado para la arquitectura*". 2. El sistema indica al servicio cuál es el documento que desea eliminar de sus estructuras 3. El sistema identifica la información del servicio asociado a ese documento. 4. El sistema procede a eliminar esa información del servicio.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No se pudo eliminar la información asociada a un documento del servicio]: se produjo un error en el momento de la modificación de las estructuras del servicio.
PRECONDICIONES
El caso de uso "*Eliminar documentos adaptados a la arquitectura*" indica cual va a ser el servicio sobre el que se va a realizar la modificación de sus estructuras. El servicio tiene que estar creado. El documento cuya información se quiere eliminar del servicio debe de existir.
POSTCONDICIONES
Se modifica la estructura del servicio eliminando la información asociada a un documento del mismo.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

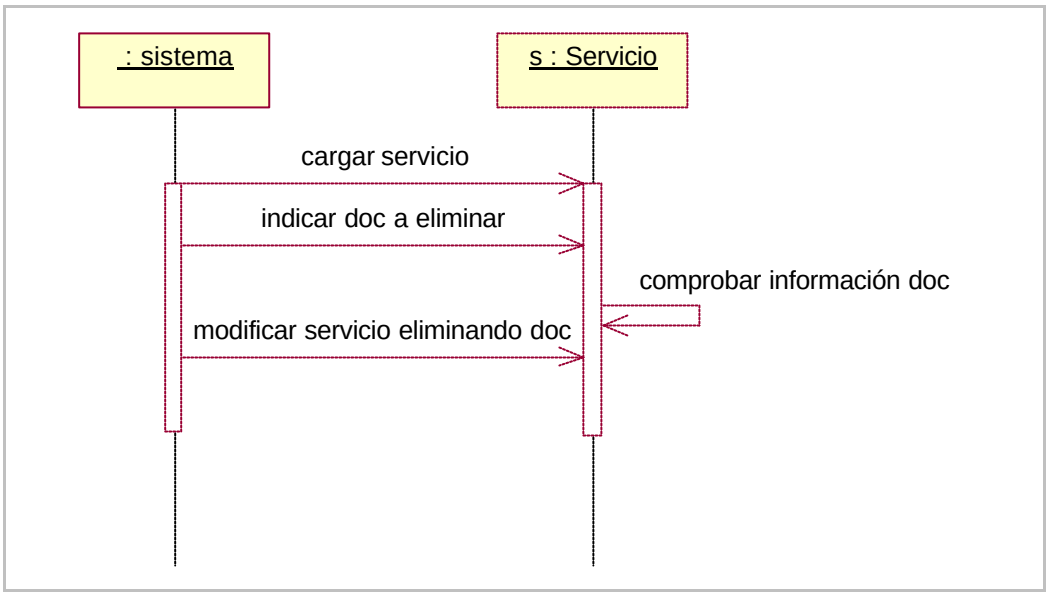


Figura 53: Análisis: Diagrama de secuencia: Eliminar un documento de un servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

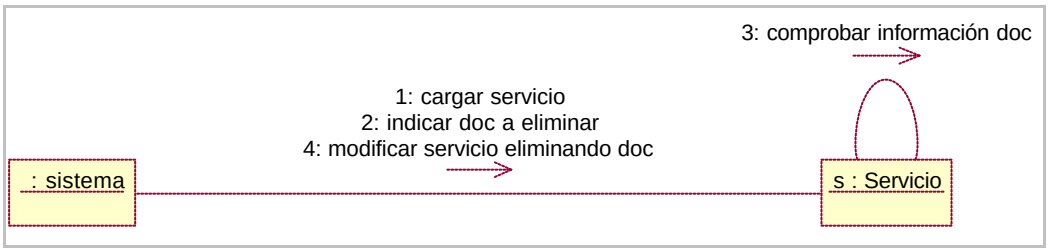


Figura 54: Análisis: Diagrama de colaboración: Eliminar un documento de un servicio sobre grupo de documentos

Escenario: Acceder y realizar búsqueda de documento mediante servicio

DESCRIPCIÓN
El usuario a través de un criterio que él establece, realizará búsquedas mediante el servicio indicado de todos los documentos que posea y que cumplan dicho criterio.
FLUJO PRINCIPAL
1. El usuario pasa al sistema un criterio de búsqueda y el servicio sobre el que se quiere realizar el acceso. 2. El sistema accede a ese servicio 3. El sistema procesa y trasforma la consulta indicada por el usuario en función de lo indicado en el fichero de configuración. 4. El servicio, en base al criterio, realiza una selección entre todos los

documentos de los que posee información agregada, identificando los que cumplen con ese criterio 5. El servicio devuelve un conjunto de resultados que consiste en los nombres de los documentos que cumplen con el criterio especificado por el usuario.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El servicio no existe]: se produjo un error porque el servicio no existe. - [Excepción: No existen documentos para ese criterio]: el servicio no devolvió ningún resultado para ese criterio.
PRECONDICIONES
No existe precondición.
POSTCONDICIONES
No existe poscondición

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

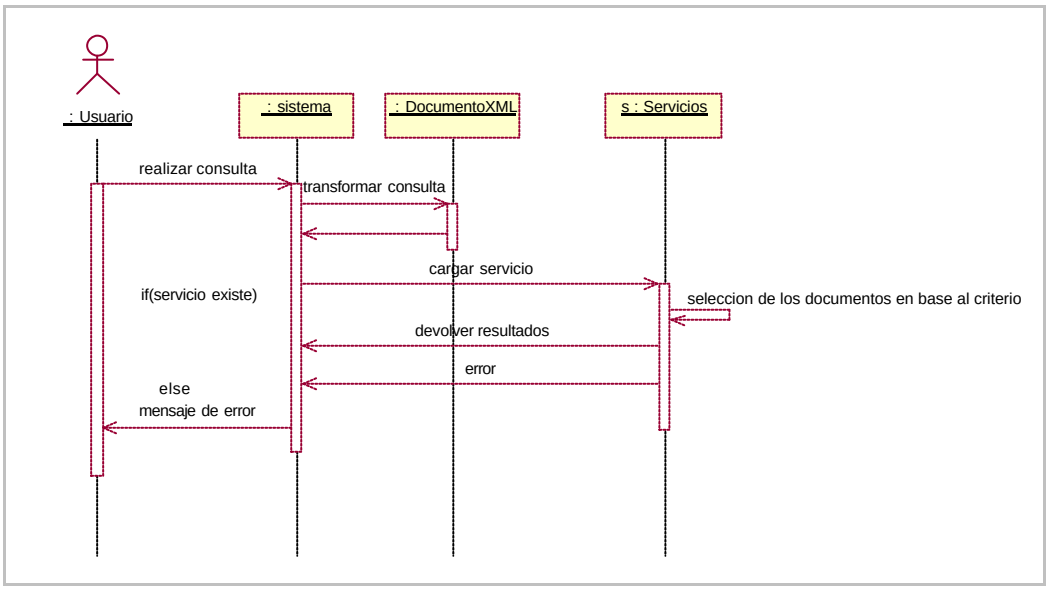


Figura 55: Análisis: Diagrama de secuencia: Acceder y realizar búsqueda de documento mediante servicio

El diagrama de colaboración de este escenario:

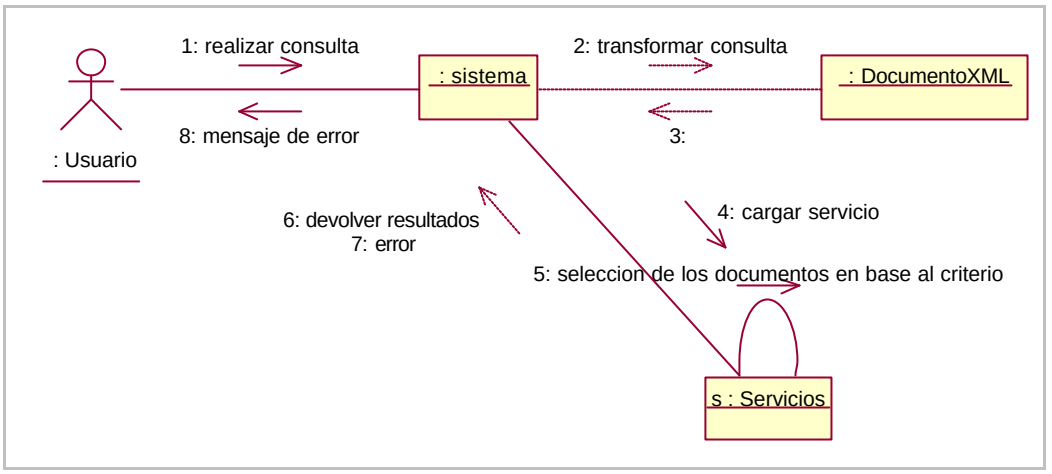


Figura 56: Análisis: Diagrama de colaboración: Acceder y realizar búsqueda de documento mediante servicio

3.1.5.4 Caso de uso: Gestionar presentaciones

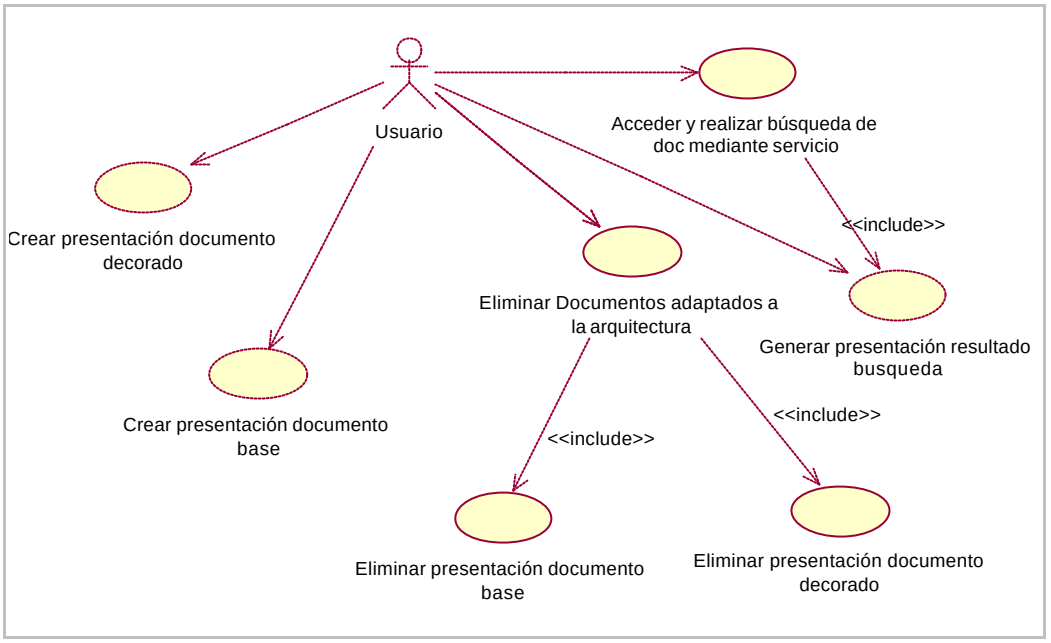


Figura 57: Análisis: Caso de uso: Gestionar presentaciones

OBJETIVOS
Este caso de uso tiene la funcionalidad de permitir crear las visualizaciones de los documentos base, de los documentos decorados así como la de crear la visualización del conjunto de documentos que devuelve el caso de uso "*Acceder y realizar búsqueda de documento mediante un servicio*", que a su vez forma parte del caso de uso general "*Gestionar Servicios*" para su posterior consulta por parte del usuario. Además permite eliminar las visualizaciones de los documentos base y de los documentos decorados. Cuando éstos son instanciados por el caso de uso general *"Eliminar Documentos adaptados a la arquitectura*", junto con el conjunto de documentos que devuelva el caso de uso que se comenta más arriba.
ACTORES
Usuario. En el caso de uso "*Gestionar Servicios*" (más concretamente el caso de uso "*Acceder y realizar búsqueda de documento mediante un servicio*") y el caso de uso "*Eliminar documentos adaptados a la arquitectura*" instancian este caso de uso general.
PRECONDICIONES
Para poder crear la presentación de un documento base es necesario que éste esté creado previamente.

Para poder crear la presentación de un documento decorado es necesario que éste esté creado previamente. Para poder presentar tanto un documento base como un documento decorado, es necesario realizar previamente una generación de la presentación de un resultado de búsqueda. Para eliminar tanto un documento base como un documento decorado, es necesario que previamente éstos se hayan creado y que además se quiera eliminar ya bien sea el documento base o el documento decorado para de ese modo eliminar también su presentación.
POSTCONDICIONES
No existen poscondiciones
FUNCIONES QUE CUMPLE
9, 10, 11, 12, 13, 18, 19

Escenario: Generar presentación de un resultado de una búsqueda

DESCRIPCIÓN
El usuario a través del caso de uso general "*Gestionar Servicios*" decide (mediante el caso de uso "*Acceder y realizar búsqueda de documentos mediante un servicio*") generar la presentación de los resultados de una consulta de acceso a un servicio obtenidos a través del mismo.
FLUJO PRINCIPAL
1. El sistema, después de que el usuario hubiese accedido, recibe el resultado de la búsqueda realizada mediante un servicio y el tipo de visualización que se quiere construir sobre ese resultado. 2. El sistema analiza el resultado que acaba de recibir 3. El sistema crea la presentación asociada al resultado que acaba de recibir 4. El usuario accede a la presentación que se acaba de generar 5. Una vez accedido, el sistema pasa a destruir la presentación que había generado previamente
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El resultado está vacío]: el resultado de la búsqueda recibido no posee ningún documento que se aproxime a lo especificado. - [Excepción: No se pudo generar la presentación]: se produjo un error en el

momento de generar la presentación del resultado.
PRECONDICIONES
Se tiene que haber accedido y realizado una búsqueda de documentos mediante un servicio previamente.
POSTCONDICIONES
No existe poscondición

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

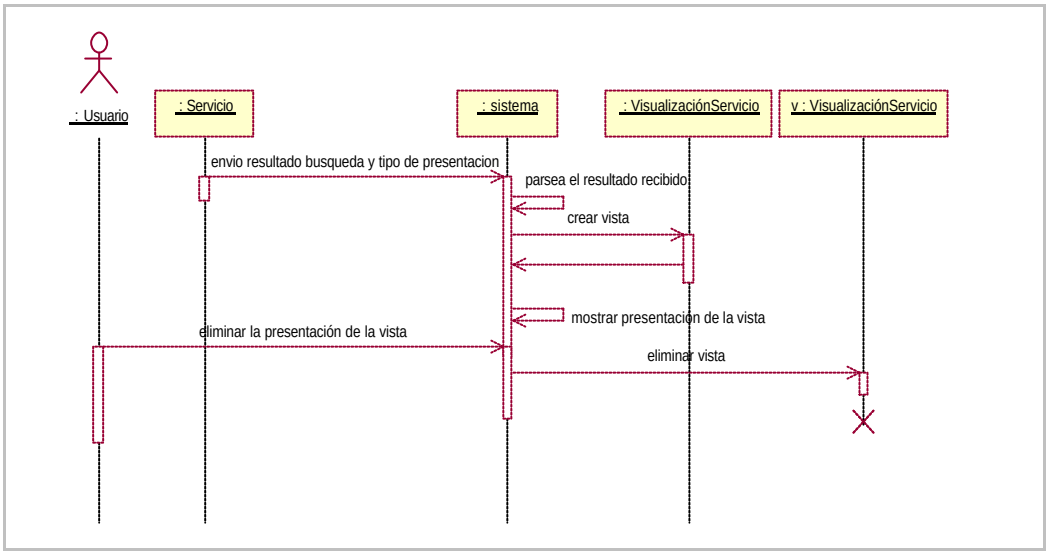


Figura 58: Análisis: Diagrama de secuencia: Generar presentación de un resultado de una búsqueda

El diagrama de colaboración de este escenario:

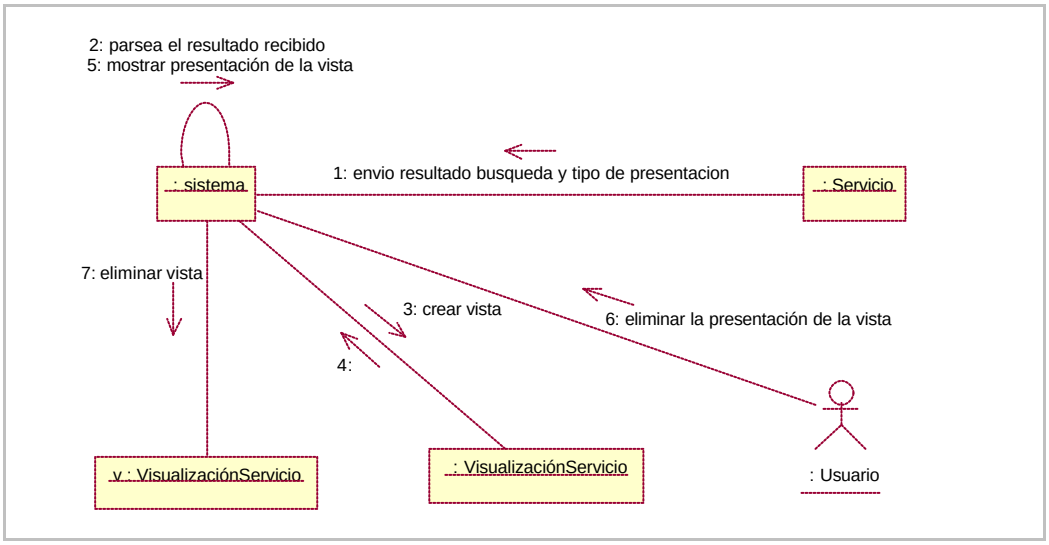


Figura 59: Análisis: Diagrama de colaboración: Generar presentación de un resultado de una búsqueda

Escenario: Crear presentación documento base

DESCRIPCIÓN
El caso de uso comienza cuando el usuario desea visualizar el contenido de un documento base.
FLUJO PRINCIPAL
1. El sistema verifica que el documento base exista 2. El sistema verifica si la presentación está ya generada 3. Si la presentación del documento base ya está generada, no hace nada 4. En caso de que no esté generada, extrae la información del documento base y genera la presentación
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento no existe]: se produjo un error porque el documento base no existe. - [Excepción: No se pudo crear la presentación asociada al documento base]: se produjo un error al crear la presentación del documento base.
PRECONDICIONES
El documento base debe de estar generado
POSTCONDICIONES
Se ha creado la presentación asociada al documento base.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

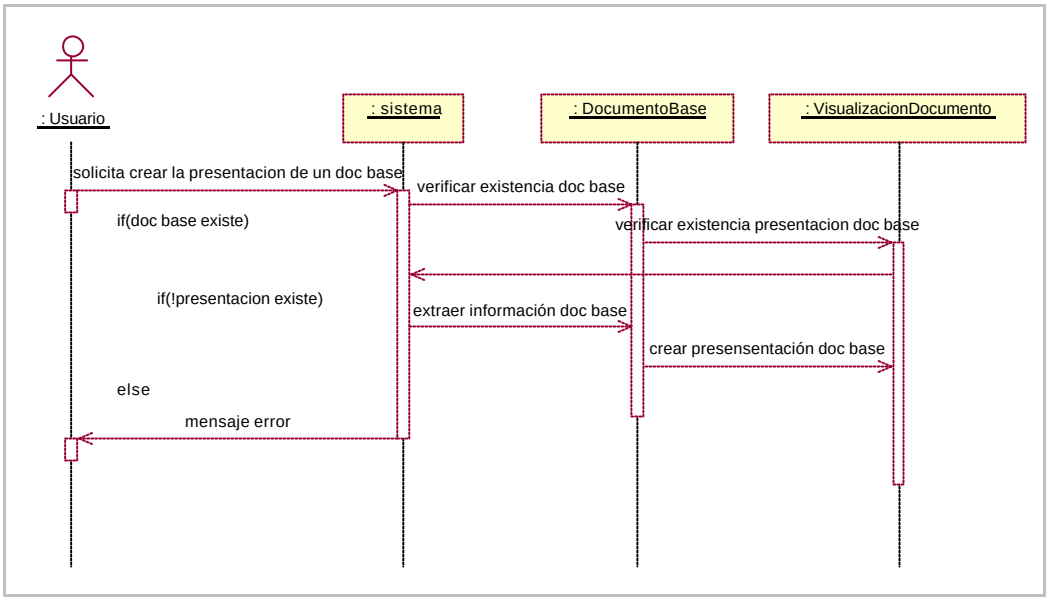


Figura 60: Análisis: Diagrama de secuencia: Crear presentación de un documento base

El diagrama de colaboración de este escenario:

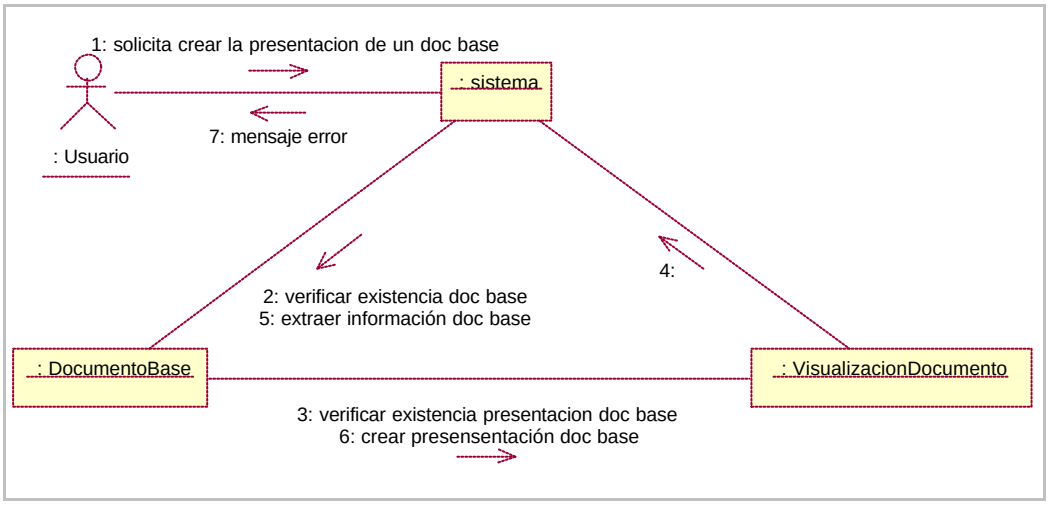


Figura 61: Análisis: Diagrama de colaboración: Crear presentación de un documento base

Escenario: Crear presentación documento decorado

DESCRIPCIÓN
El caso de uso comienza cuando el usuario desea visualizar el contenido de un documento decorado.
FLUJO PRINCIPAL
1. El sistema verifica que el documento decorado exista 2. El sistema verifica si la presentación está ya generada

3. Si la presentación del documento decorado ya está generada no hace nada 4. En caso de que no esté generada, extrae la información del documento decorado y genera la presentación
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento no existe]: se produjo un error porque el documento decorado no existe. - [Excepción: No se pudo crear la presentación asociada al documento decorado]: se produjo un error al crear la presentación del documento decorado.
PRECONDICIONES
El documento decorado debe de estar generado
POSTCONDICIONES
Se ha creado la presentación asociada al documento decorado.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

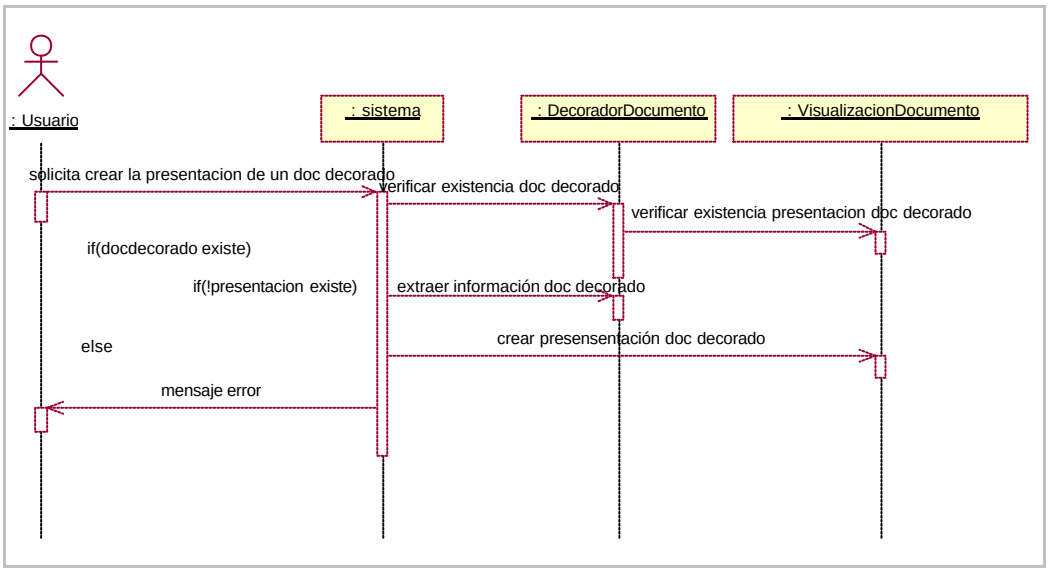


Figura 62: Análisis: Diagrama de secuencia: Crear presentación de un documento decorado

El diagrama de colaboración de este escenario:

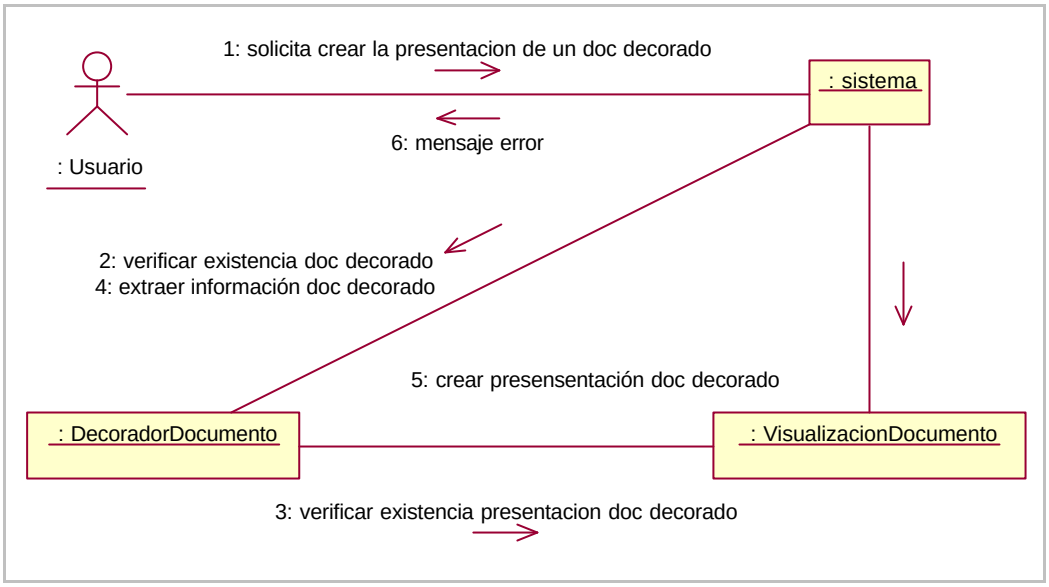


Figura 63: Análisis: Diagrama de colaboración: Crear presentación de un documento decorado

Escenario: Eliminar presentación documento base

DESCRIPCIÓN
Este caso de uso comienza cuando el usuario decide eliminar un documento base. Por tanto quien va a iniciar este caso de uso es *"Eliminar Documentos adaptados a la arquitectura*".
FLUJO PRINCIPAL
1. El sistema recibe una indicación de cual es el documento base sobre el que se tiene que eliminar la presentación. 2. El sistema verifica que exista el documento base. 3. Si el documento base existe, verifica que existe la presentación asociada a ese documento. 4. Procede a la eliminación de la presentación del documento base. 5. En caso de que la presentación no exista, entonces no hace nada
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento base no existe]: se produjo un error porque el documento base en cuestión no existe.
PRECONDICIONES
El documento base tiene que estar creado previamente y no se puede eliminar

hasta que no se haya eliminado la presentación
POSTCONDICIONES
La presentación del documento base se ha eliminado

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

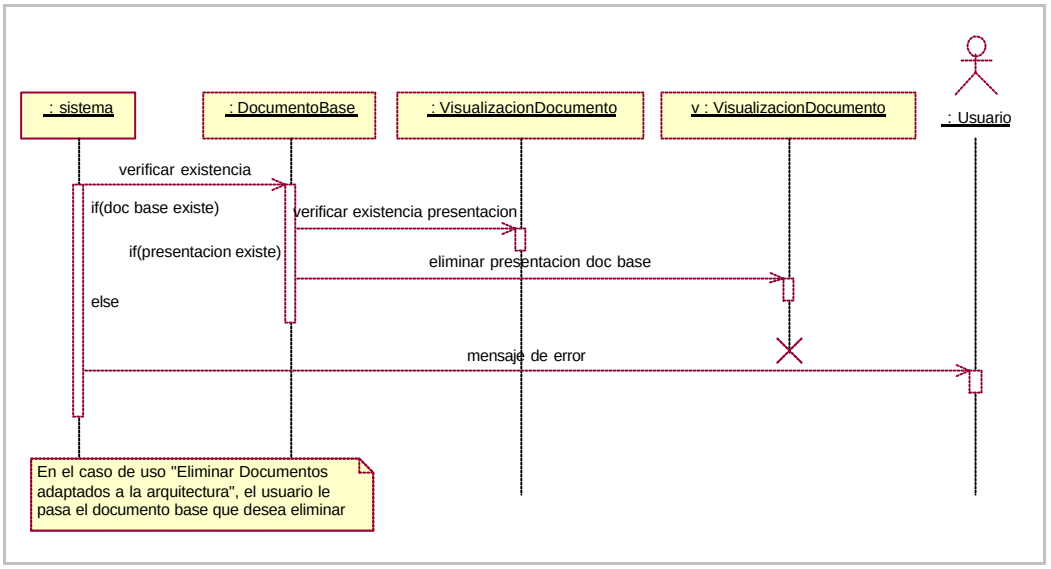


Figura 64: Análisis: Diagrama de secuencia: Eliminar la presentación del documento base

El diagrama de colaboración de este escenario:

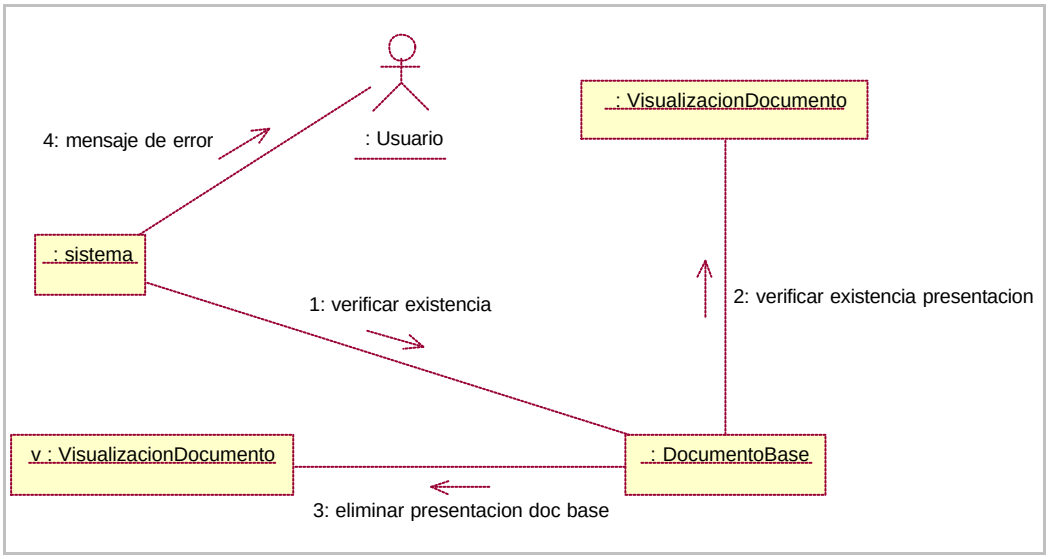


Figura 65: Análisis: Diagrama de colaboración: Eliminar la presentación del documento base

Escenario: Eliminar la presentación de un documento decorado

DESCRIPCIÓN
Este caso de uso comienza cuando el usuario decide eliminar un documento base, y a su vez se eliminan los documentos decorados asociados. Por tanto quien va a iniciar este caso de uso es *"Eliminar Documentos adaptados a la arquitectura*".
FLUJO PRINCIPAL
1. El sistema recibe una indicación de cual es el documento decorado sobre el que se tiene que eliminar la presentación. 2. El sistema verifica que exista el documento decorado. 3. Si el documento decorado existe, verifica que existe la presentación asociada a ese documento. 4. Procede a la eliminación de la presentación del documento decorado. 5. En caso de que la presentación no exista, entonces no hace nada
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: El documento decorado no existe]: se produjo un error porque el documento decorado en cuestión no existe.
PRECONDICIONES
El documento decorado tiene que estar creado previamente y no se puede eliminar hasta que no se haya eliminado la presentación
POSTCONDICIONES
La presentación del documento decorado se ha eliminado

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

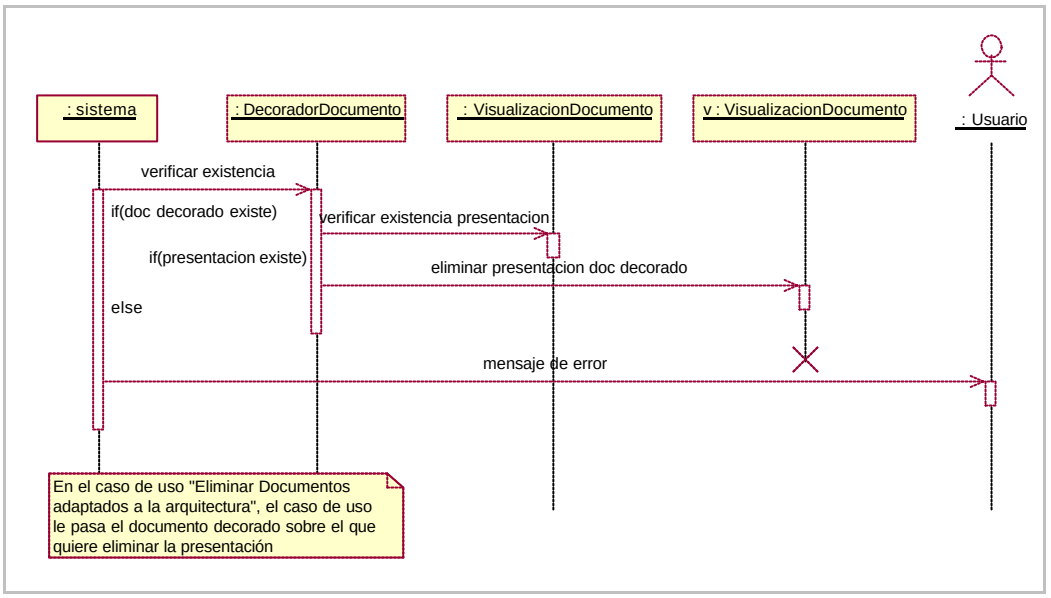


Figura 66: Análisis: Diagrama de secuencia: Eliminar la presentación de un documento decorado

El diagrama de colaboración de este escenario:

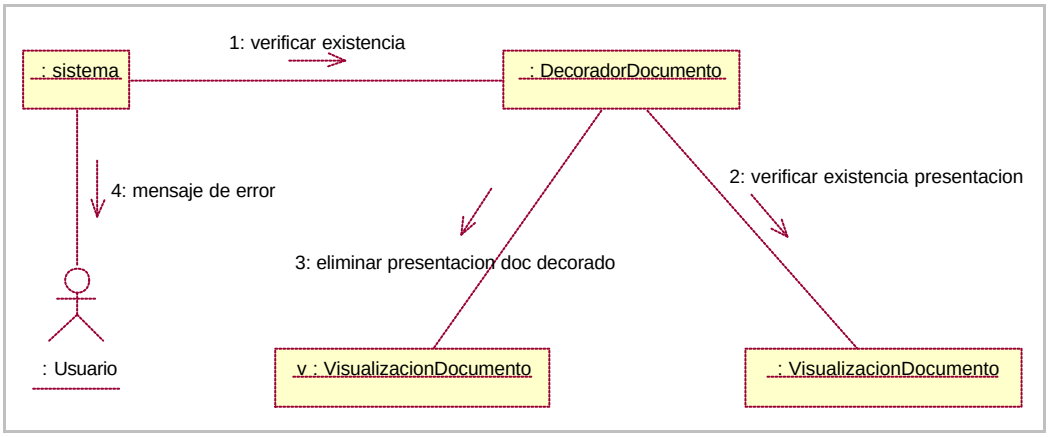


Figura 67: Análisis: Diagrama de colaboración: Eliminar la presentación de un documento decorado

3.2 Diseño de la arquitectura

En esta sección se detalla el diseño del sistema, cuyo objetivo es mostrar cómo se resuelve el problema planteado en la especificación de requisitos.

3.2.1 Diagrama de clases

El diagrama de clase que se desarrolló es el siguiente:

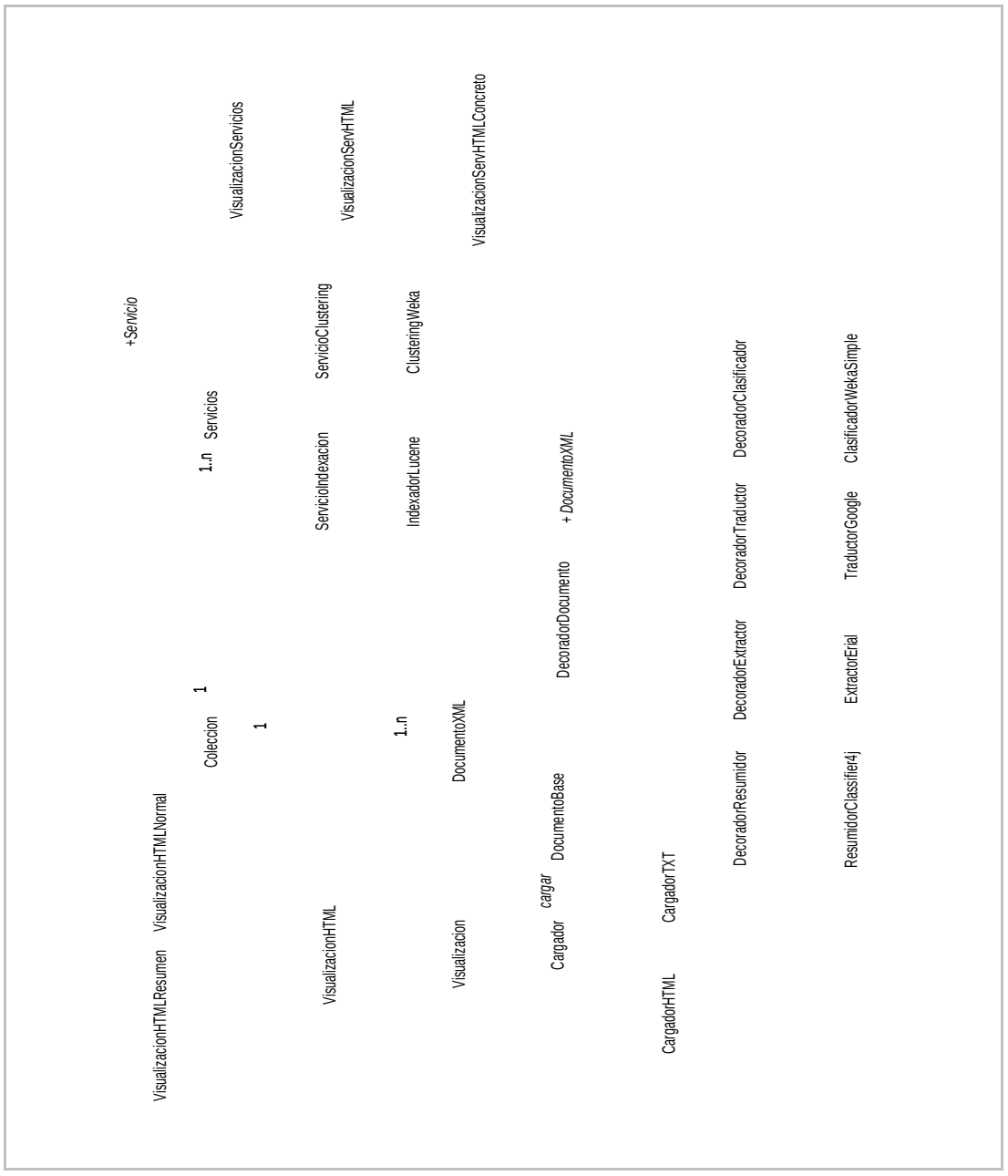


Figura 68: Diseño: Diagrama de clases

Como se puede observar en la figura anterior, se han realizado ciertos cambios y ampliaciones con respecto al diagrama de clases de la fase de análisis. A continuación se enumeran estas diferencias:

- La clase *Vision* pasa a llamarse *DocumentoXML,* ya que como se verá en el capítulo de "Implementación y pruebas", el formato usado para el almacenamiento de los documentos, tanto los documentos base como decorados, está basado en el uso XML.
- La clase *DecoradorDocumento* se divide ahora en 4 tipos, que serán subclases de la primera. Éstas se corresponden con 4 servicios básicos de sobre documentos individuales que se van a proporcionar en la arquitectura que aquí se desarrolla. En concreto tendremos, la clase *DecoradorResumidor*: encargado de generar resúmenes; la clase *DecoradorExtractor:* encargado de extraer términos[1] a partir del texto del documento; la clase *DecoradorTraductor:* encargado de generar la traducción a un determinado idioma y la clase *DecoradorClasificador:* encargado de generar la clasificación de un documento.
- La clase *Servicios* también se divide en 2 tipos, que serán subclases de la primera. Éstas son los 2 tipos de servicios sobre grupos de documentos que se proporcionan en la arquitectura, como son: la clase *ServicioIndexador* y la clase *ServicioClustering*.
- Además de todo lo expuesto anteriormente, el diagrama de clases que se presenta incluye clases que ofrecen implementaciones concretas para cada una de las clases que se acaban de detallar. En concreto, tenemos las siguientes clases:
  - *ResumidorClasssifier4j*, *ExtractorErial*, *TraductorGoogle*, *ClasificadorWekaSimple*
  - *IndexadorLucene*, *ClusterWekaSimple*
  - *ConvertHTML2XML*, *ConverTXT2XML*
  - *VisualizacionHTMLResumen*, *VisualizacionHTMLNormal*
  - *VisualizacionServHTMLConcreto*

En el diagrama que se presenta en la figura anterior, no se han representado ciertas clases que son secundarias en la arquitectura, ya que esto dificultaría el entendimiento del sistema. De todos modos se incluyen a continuación para de esa forma conseguir un entendimiento total del sistema.

[1] Término se refiere a "palabra clave" o secuencia de palabras útil, por ejemplo para tareas de indexación o clasificación.

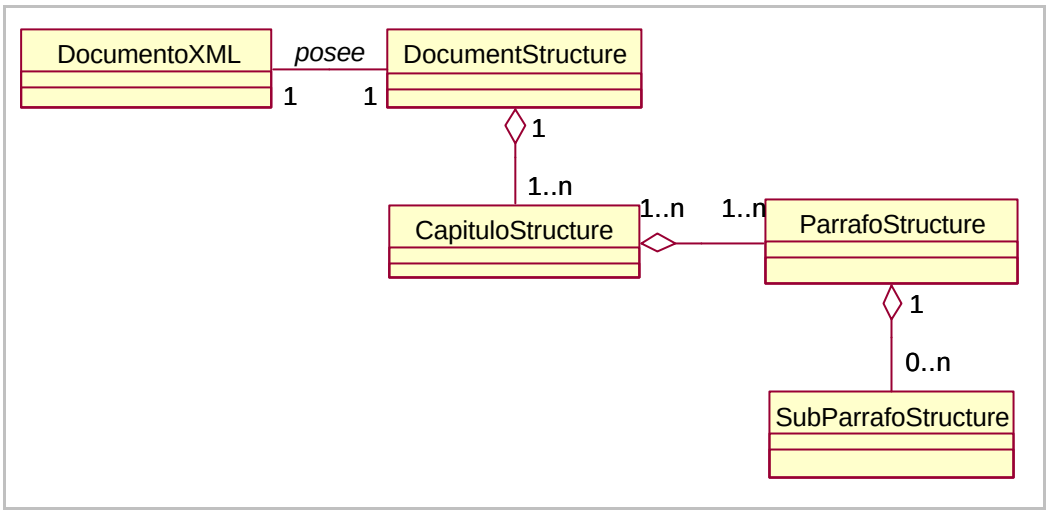


Figura 69: Diseño: Diagrama de clases (continuación)

El diagrama de clases que aquí se muestra, representa la estructura de un documento base, es decir, una vez extraída y organizada la información del documento base original, se deberá devolver una estructura del tipo *DocumentStructure* que a su vez está formada por uno o varias estructuras del tipo *CapituloStructure*, a su vez, formado por una o varias estructuras del tipo *ParrafoStructure,* que finalmente pueden estar formadas por una estructura del tipo *SubParrafoStructure*. Todas estas clases se describen detalladamente en el diccionario de clases que se encuentra más adelante.

Es necesario destacar que en el diseño de la arquitectura se han tomado como punto de partida ciertos patrones de diseño como son el patrón **Decorador**, el patrón **Cadena de Responsabilidad** y el patrón **Estrategia** [10]. Un documento decorado puede proceder tanto de un documento base como de un documento ya decorado mediante el uso del patrón *Decorador*. Lo mismo ocurre con el encadenamiento de servicios sobre grupos de documentos. No existe una clase propiamente dicha que realice esa cadena sino que de eso se encarga el propio patrón *Cadena de responsabilidades*. En el caso de las visualizaciones, se puede aplicar el patrón Estrategia de modo que la implementación que se escoja para las presentaciones sea independiente del objeto que lo use. La ventaja del uso de patrones es que éstos describen un problema que ocurre una y otra vez en el entorno, haciendo que sea más fácil reutilizar buenos diseños y arquitecturas. A continuación se van a detallar las características de cada uno de ellos.

3.2.1.1 Propósito del patrón decorador

"*El propósito fundamental del patrón decorador es asignar responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la herencia para extender la funcionalidad*"

Erich Gamma – Addison Wesley

En algunos casos queremos añadir responsabilidades a objetos individuales en vez de a toda una clase. Un modo de añadir responsabilidades es a través de la herencia. Sin embargo esto no es flexible, ya que la asignación se hace

estáticamente. Por tanto la solución es encerrar el objeto sobre el que queremos aplicar una nueva funcionalidad dentro de otro que justamente lo haga. Al objeto confinante se le llama decorador. El decorador se ajusta a la interfaz del componente que decora de manera que su presencia es transparente. El decorador reenvía las peticiones al componente y puede realizar acciones adicionales antes o después del reenvío. Dicha transparencia permite anidar decoradores recursivamente, permitiendo de este modo un número ilimitado de responsabilidades añadidas.

En el caso que nos ocupa, el objeto componente que encierra toda la funcionalidad no es más que la clase *DocumentoXML* y el objeto decorador que se encarga de realizar las decoraciones es la clase *DecoradorDocumento* que a su vez está compuesto de subclases hijas que son todos aquellos decoradores que proporcionan algún tipo de servicio sobre documentos individuales. La clase *DecoradorDocumento* mantiene una referencia al objeto *DocumentoXML*.

El objeto sobre el que se pueden asignar responsabilidades adicionales y que se podría considerar como caso base en la recursividad es la descrita por la clase *DocumentoBase*.

3.2.1.2 Propósito del patrón Cadena de responsabilidad

"*El propósito fundamental del patrón cadena de responsabilidad es el evitar acoplar el emisor de una petición a su receptor, dando a más de un objeto la posibilidad de responder a la petición. De este modo encadena los objetos receptores y pasa la petición a través de la cadena hasta que es procesada por algún objetos*"

Erich Gamma – Addison Wesley

La idea es que el objeto de la cadena recibe la petición y o bien la procesa o bien la redirige al siguiente candidato de la cadena, el cual hace lo mismo. El objeto que hizo la petición no tiene un conocimiento de quien la tratará, por lo que se dice que la petición tiene un receptor implícito. Para reenviar la petición a lo largo de la cadena y para garantizar que los receptores permanecen implícitos, cada objeto de la cadena comparte una interfaz común para procesar las peticiones y para acceder a su sucesor en la cadena.

En el caso que aquí se trata, la clase *Servicios* define la clase para tratar las peticiones. Las clases *ServicioIndexacion* y *ServicioClustering* tratan las peticiones indicadas por el usuario. En el caso de la cadena de servicios, es la propia clase *Servicios* la encargada de gestionar cuales van a ser los pasos seguidos hasta llegar al servicio final estableciendo la secuencia a seguir a partir de cada uno de los sucesores.

3.2.1.3 Propósito del patrón Estrategia

"*Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que un algoritmo varía independientemente de los clientes que lo usen.*"

Erich Gamma – Addison Wesley

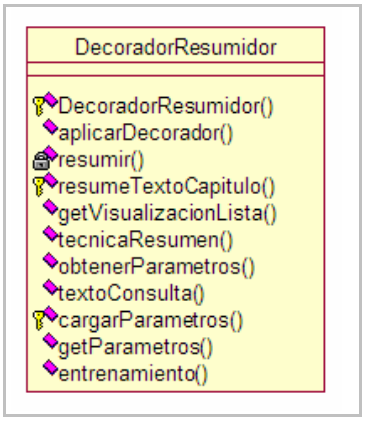
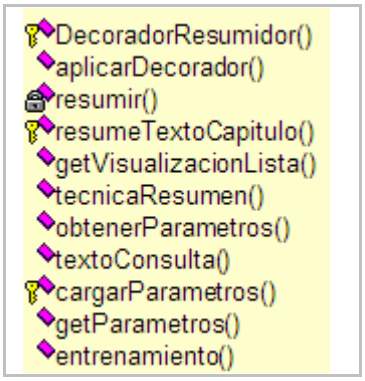
La idea de partida es que podrán existir muchos tipos de visualizaciones que se pueden aplicar tanto sobre un documento como sobre el resultado proporcionado por un servicio tras una consulta. Codificar esas presentaciones en las clases que lo usan no resulta una buena práctica por diversos motivos:

- Si se tiene que incluir en la propia clase la visualización, ésta se vuelve más compleja si se tiene que incluir dicho código sobre todo si se permiten diversas presentaciones para una misma clase.
- Cada una de las presentaciones será apropiada en determinado momento. No se tiene por que permitir múltiples algoritmos si no se van a usar.

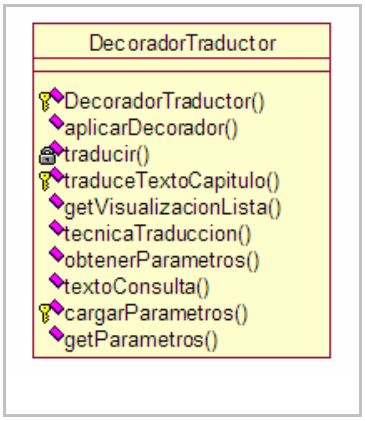
Estos problemas se pueden evitar definiendo clases que encapsulen las diferentes presentaciones. E nuestro caso nos hemos centarado en una presentación basada en HMTL. Se ha optado por delegar la presentación de los documentos, tanto documentos base como sus decoraciones, tanto de forma aislada como cuano participan en algún servicio de grupo de documentos, usando respectivamente las clases VisualizacionHTML y VisualizacionServHMTL, que son las responsables de gestionar los detalles de patrón Estrategia.

3.2.2 Diccionario de clases

Llegados a este punto, es necesario revisar las clases que se incorporaron en la fase de análisis para realizar mejoras. A continuación se procede a detallar esas clases que no incluyeron en la etapa anterior. Es necesario indicar que se va a hacer uso de un fichero de configuración en el que se incluirá los nombres de los servicios sobre documentos individuales, así como todos los parámetros necesarios, como pueden ser la clase que implementa determinado servicio, la fuente sobre la que se va a aplicar el servicio, la clase que implementa la presentación del documento y cada uno de los parámetros que son propios de cada uno de las implementaciones que se hayan desarrollado.

CLASE DECORADOR RESUMIDOR	
 **Figura 70:** Diseño: Clase DecoradorResumidor	Clase abstracta que representa las decoraciones realizadas exclusivamente por servicios de documentos individuales que son resumidores.
ATRIBUTOS	
	Los atributos que posee son heredados de la superclase *DecoradorDocumento*
MÉTODOS	
 **Figura 71:** Diseño: Métodos clase DecoradorResumidor	DecoradorResumidor: es el constructor de esta clase. Es el encargado de cargar los parámetros, de entrenar el resumidor, en caso de que sea necesario, y de aplicar el decorador el documento que se le indique. aplicarDecorador: método heredado de la superclase encargado en este caso de llamar al método resumir. resumir: encargado de asignar al atributo heredado resultado, el resumen ya bien sea de un documento base ya bien sea de un documento decorado. resumeTextoCapitulo: es el encargado de que para cada capitulo se realice el resumen devolviendo un objeto de tipo *DecoracionStructure*. getVisualizacionLista: encargado de devolver la visualización que se le indica como parámetros en el fichero de configuración. tecnicaResumen: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un String devolverá su resumen. obtenerParametros: método abstracto que tiene que implementar el usuario indicando los nombres de los

	Parámetros que va a usar. textoConsulta: método abstracto que tiene que implementar el usuario y que permite que cuando se realicen búsquedas, tomando como base el resultado de este resumidor la consulta se transforme según lo considere el usuario. cargarParametros: método abstracto que permite cargar los parámetros asociados al resumidor concreto desde el fichero de configuración. entrenamiento: método abstracto que tiene que implementar el usuario para realizar el entrenamiento sobre el resumidor concreto, en caso de que el generador de resúmenes utilizado lo necesite.

CLASE DECORADOR TRADUCTOR	
 **Figura 72:** Diseño: Clase DecoradorTraductor	Clase que representa las decoraciones realizadas exclusivamente por servicios de documentos individuales que son traductores.
ATRIBUTOS	
	Los atributos que posee son heredados de la superclase *DecoradorDocumento*.
MÉTODOS	
	DecoradorTraductor: es el constructor de esta clase. Es el encargado de cargar los parámetros y de aplicar el decorador sobre lo que se le pase. aplicarDecorador: método heredado de la superclase encargado en este caso de llamar al método traducir. traducir: encargado de asignar al atributo heredado

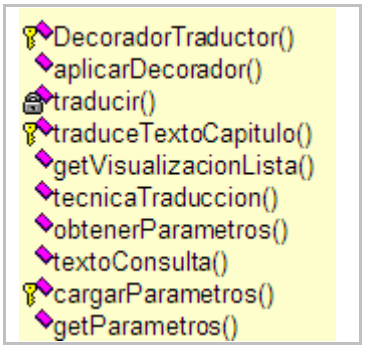


Figura 73: Diseño: Métodos clase DecoradorTraductor

resultado, la traducción ya bien sea de un documento base ya bien sea de un documento decorado.

traduceTextoCapitulo : es el encargado de que para cada capitulo se realice la traducción devolviendo un objeto de tipo *DecoracionStructure*.

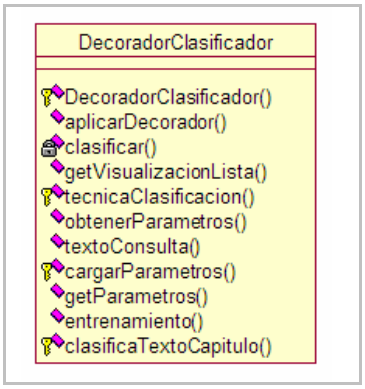
getVisualizacionLista: encargado de devolver la visualización que se le indica como parámetros en el fichero de configuración.

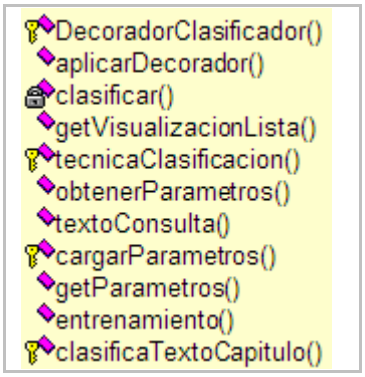
tecnicaTraducción: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un String devolverá su traducción.

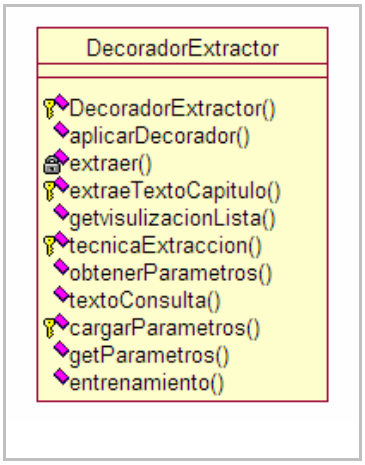
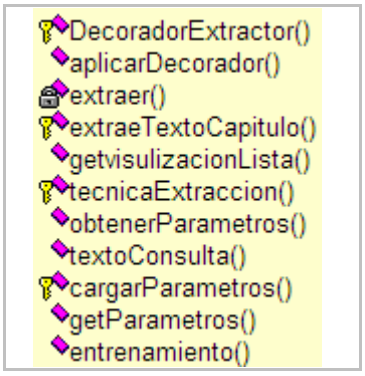
obtenerParametros: método abstracto que tiene que implementar el usuario indicando los nombres de los Parámetros que va a usar.

textoConsulta: método abstracto que tiene que implementar el usuario y que permite que cuando se realicen búsquedas, la consulta se transforme según lo considere el usuario.

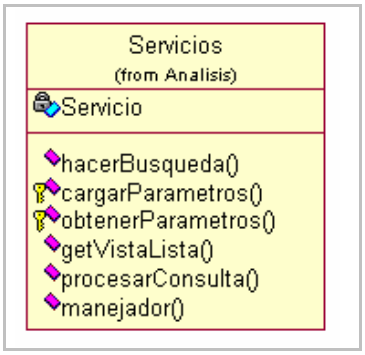
cargarParametros: método abstracto que permite cargar los parámetros asociados al traductor concreto del fichero de configuración.

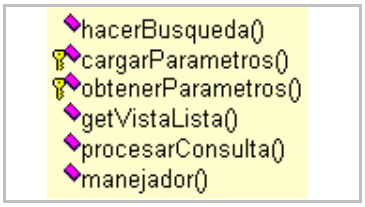
CLASE DECORADOR CLASIFICADOR	
 **Figura 74:** Diseño: Clase DecoradorClasificador	Clase que representa las decoraciones realizadas exclusivamente por servicios de documentos individuales que son clasificadores.
ATRIBUTOS	
	Los atributos que posee son heredados de la superclase *DecoradorDocumento*.

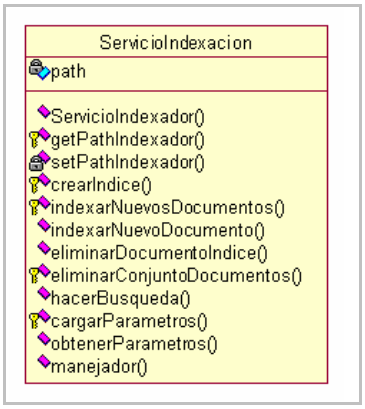
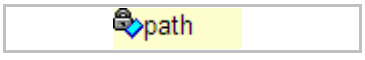
MÉTODOS	
 **Figura 75:** Diseño: Métodos clase DecoradorClasificador	DecoradorClasificador: es el constructor de esta clase. Es el encargado de cargar los parámetros, entrenar y de aplicar el decorador sobre lo que se le pase. aplicarDecorador: método heredado de la superclase encargado en este caso de llamar al método clasificar. clasificar: encargado de asignar al atributo heredado resultado, la clasificación ya bien sea de un documento base ya bien sea de un documento decorado. clasificaTextoCapitulo: es el encargado de que para cada capitulo se realice la clasificación devolviendo un objeto de tipo *DecoracionStructure*. getVisualizacionLista: encargado de devolver la visualización que se le indica como parámetros en el fichero de configuración. tecnicaClasificacion: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un String devolverá su clasificación. obtenerParametros: método abstracto que tiene que implementar el usuario indicando los nombres de los Parámetros que va a usar. textoConsulta: método abstracto que tiene que implementar el usuario y que permite que cuando se realicen búsquedas, la consulta se transforme según lo considere el usuario. cargarParametros: método abstracto que permite cargar los parámetros asociados al clasificador concreto del fichero de configuración.

CLASE DECORADOR EXTRACTOR	
 **Figura 76:** Diseño: Clase DecoradorExtractor	Clase que representa las decoraciones realizadas exclusivamente por servicios de documentos individuales que son extractores.
ATRIBUTOS	
	Los atributos que posee son heredados de la superclase *DecoradorDocumento*.
MÉTODOS	
 **Figura 77:** Diseño: Métodos clase DecoradorExtractor	DecoradorExtractor: es el constructor de esta clase. Es el encargado de cargar los parámetros, entrenar y de aplicar el decorador sobre lo que se le pase. aplicarDecorador: método heredado de la superclase encargado en este caso de llamar al método extraer. extraer: encargado de asignar al atributo heredado resultado, la extracción ya bien sea de un documento base ya bien sea de un documento decorado. extraeTextoCapitulo: es el encargado de que para cada capitulo se realice la clasificación devolviendo un objeto de tipo *DecoracionStructure*. getVisualizacionLista: encargado de devolver la visualización que se le indica como parámetros en el fichero de configuración. tecnicaExtracción: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un String devolverá la lista de términos extraídos a partir del mismo. obtenerParametros: método abstracto que tiene que

	implementar el usuario indicando los nombres de los Parámetros que va a usar. textoConsulta: método abstracto que tiene que implementar el usuario y que permite que cuando se realicen búsquedas, la consulta se transforme según lo considere el usuario. cargarParametros: método abstracto que permite cargar los parámetros asociados al extractor concreto del fichero de configuración.

CLASE SERVICIO	
 **Figura 78:** Diseño: Clase Servicio	Clase que representa los servicios sobre grupos de documentos que va a proporcionar la arquitectura.
ATRIBUTOS	
 **Figura 79:** Diseño: Clase Servicio	Este atributo representa el enlace al sucesor de la tarea de servicio en los pasos de servicio de cadena.
MÉTODOS	

 **Figura 80:** Diseño: Métodos Clase Servicios	cargarParametros: método abstracto que carga los parámetros de las implementaciones concretas obtenerParametros: método abstracto que permite obtener el nombre de los parámetros que usa la implementación concreta del servicio procesarConsulta: método abstracto que permite procesar la consulta que se realiza sobre el servicio hacerBusqueda: método abstracto que realiza la búsqueda de documentos mediante una consulta a través de un servicio manejador: método encargado de manejar las peticiones de cadenas de servicios

CLASE SERVICIO INDEXADOR	
 **Figura 81:** Diseño: Clase ServicioIndexador	Clase que representa los servicios sobre grupos de documentos que son indexadores.
ATRIBUTOS	
 **Figura 82:** Diseño: Atributos clase ServicioIndexador	Path es la ruta donde se encuentra este servicio de indexación, donde el servicio de indexación gestionará las estrujas que necesite para su funcionamiento, como por ejemplo los índices, etc.
MÉTODOS	
	ServicioIndexador: es el constructor de esta clase. Es el encargado de cargar los parámetros, crear el índice y de indexar los nuevos documentos. getPathIndexador: devuelve la ruta del servicio

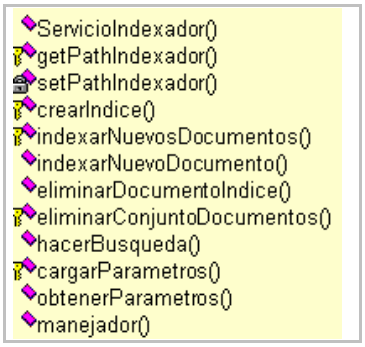


Figura 83: Diseño: Métodos clase ServicioIndexador

setPathIndexador: asigna la ruta del servicio.

crearIndice: es un método abstracto que tiene que implementar el usuario incluyendo el código pertinente para crear el índice del indexador.

indexarNuevosDocumentos: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un grupo de *DocumentoXML*, llamará a *indexarNuevoDocumento* para añadir cada uno por separado.

indexarNuevoDocumento: es un método abstracto que tendrá que ser implementado por el usuario en su clase. Será el que pasándole un *DocumentoXML*, lo añadirá al índice creado previamente.

eliminarDocumentoIndice: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole *DocumentoXML*, lo eliminará del índice.

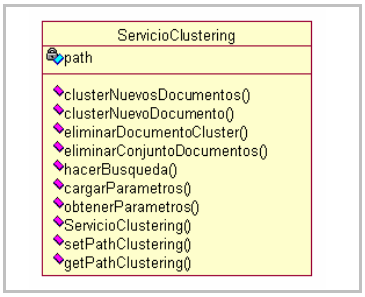
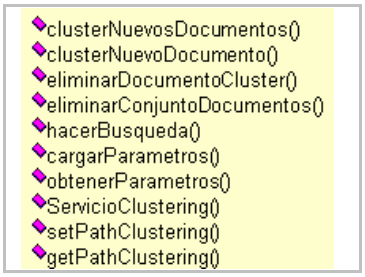
eliminarConjuntoDocumentos: es un método abstracto que tendrá que ser implementado por el usuario en su clase. Será el que pasándole un grupo de *DocumentoXML*, llamará a *eliminarDocumentoIndice* para eliminar cada uno por separado.

hacerBusqueda: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Incluirá el código encargado de realizar la consulta que le pasa el usuario sobre el índice.

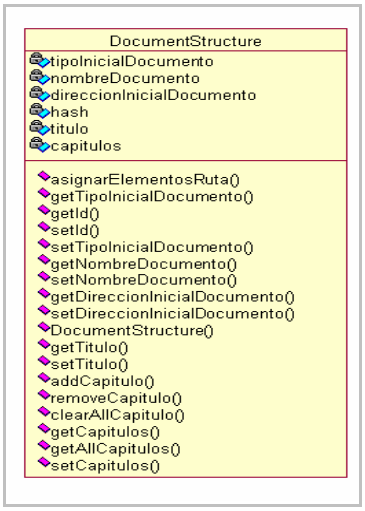
obtenerParametros: método abstracto que tiene que implementar el usuario indicando los nombres de los Parámetros que va a usar.

cargarParametros: método abstracto que permite cargar los parámetros asociados al indexador concreto del fichero de configuración.

manejador: método encargado de manejar las peticiones de cadenas de servicios

CLASE SERVICIO CLUSTERINGS	
 **Figura 84:** Diseño: Clase ServicioClusterings	Clase que representa los servicios sobre grupos de documentos que son clusterings.
ATRIBUTOS	
 **Figura 85:** Diseño: Atributos clase ServicioClusterings	Path es la ruta donde se encuentra este servicio de clusterings.
MÉTODOS	
 **Figura 86:** Diseño: Métodos clase ServicioClusterings	ServicioClusterings es el constructor de esta clase. Es el encargado de cargar los parámetros, crear el índice y de indexar los nuevos documentos. getPathClusterings: devuelve la ruta del servicio setPathClusterings: asigna la ruta del servicio. clusterNuevosDocumentos: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un grupo de *DocumentoXML*, llamará a *clusterNuevoDocumento* para añadirlos cada uno por separado. clusterNuevoDocumento: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un *DocumentoXML*, se añadirá al a la estructura creado previamente por el servicio. eliminarDocumentoCluster: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole *DocumentoXML*, lo eliminará del cluster. eliminarConjuntoDocumentos: es un método abstracto

	que tendrá que ser implementado por el usuario en sus clase. Será el que pasándole un grupo de *DocumentoXML*, llamará a *eliminarDocumentoCluster* para eliminarlos cada uno por separado. hacerBusqueda: es un método abstracto que tendrá que ser implementado por el usuario en sus clase. Será el que realice la consulta sobre la estructura del servicio que le pasa el usuario. obtenerParametros: método abstracto que tiene que implementar el usuario indicando los nombres de los Parámetros que va a usar. cargarParametros: método abstracto que permite cargar los parámetros asociados al indexador concreto del fichero de configuración. manejador: método encargado de manejar las peticiones de cadenas de servicios

CLASE DOCUMENTSTRUCTURE	
 **Figura 87:** Diseño: Clase DocumentStructure	Clase que representa la estructura de un documento original
ATRIBUTOS	
	TipoInicialDocumento: es la extensión del documento original NombreDocumento: es el nombre del documento original direccionInicialDocumento: es la ruta del documento

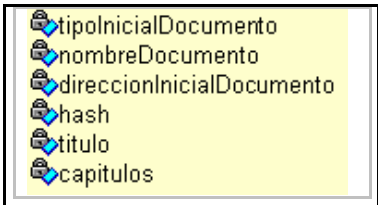


Figura 88: Diseño: Atributos clase DocumentStructure

original

hash: es el identificador del documento original

titulo: es el titulo del documento original

capitulos: es la lista de capítulos del documento original

MÉTODOS

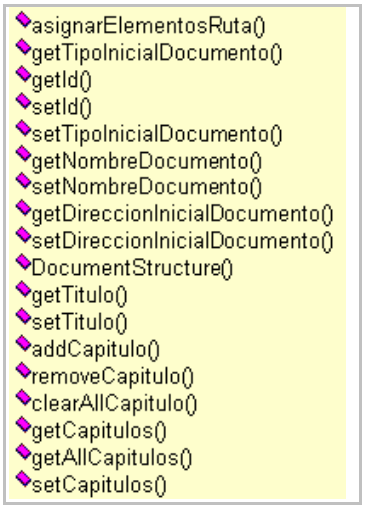


Figura 89: Diseño: Métodos clase DocumentStructure

asignarElementosRuta: este método recibe el nombre del documento original, su dirección y su identificador y se los asigna además del tipo de documento.

getTipoInicialDocumento: devuelve la extensión del documento original

getId: devuelve el identificador del documento

setId: asigna el identificador del documento

setTipoInicialDocumento: asigna al atributo la extensión del documento original

getNombreDocumento: devuelve el nombre del documento original

setNombreDocumento: asigna al atributo el nombre del documento original

getDireccionInicialDocumento: devuelve la ruta del documento original

setdireccionInicialDocumento: asigna al atributo la dirección del documento original.

DocumentStructure: es el constructor de la clase

getTitulo: devuelve el titulo del documento

setTitulo: asigna el titulo al atributo

addCapitulo: añade un elemento a la lista de capítulos

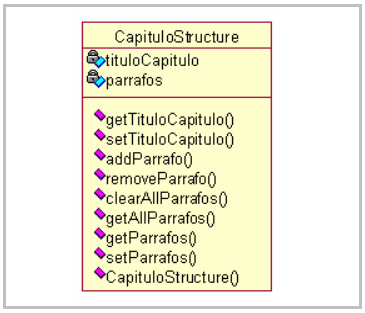
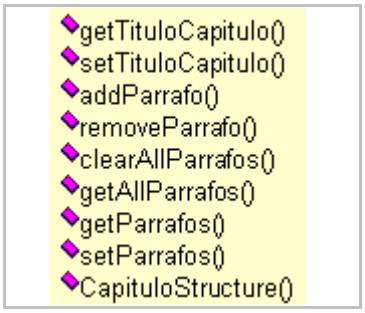
removeCapitulo: elimina un elemento de la lista de capitulos

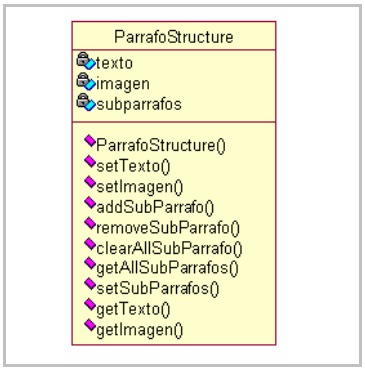
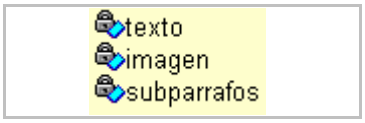
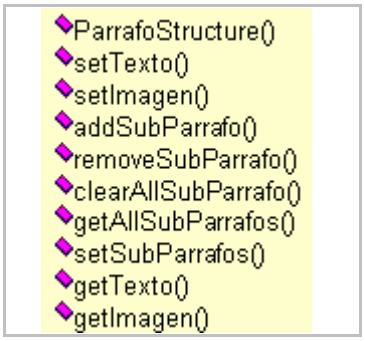
clearAllCapitulo: elimina todos los capítulos de la lista

getCapitulos: devuelve la lista de capítulos

getAllCapitulos: devuelve un iterador de la lista de capítulos

setCapitulos: asigna una lista al atributo capítulos

CLASE CAPITULOSTRUCTURE	
 **Figura 90:** Diseño: Clase CapituloStructure	Clase que representa la estructura de un capitulo del documento original.
ATRIBUTOS	
 **Figura 91:** Diseño: Atributos CapituloStructure	tituloCapitulo: es el titulo que posee ese capitulo en caso de existir parrafos: es la lista de parrafos que posee ese capitulo.
MÉTODOS	
 **Figura 92:** Diseño: Métodos CapituloStructure	getTituloCapitulo: devuelve el titulo del capitulo setTituloCapitulo: asigna al atributo el titulo del capitulo addParrafo: añade un parrafo a la lista de parrafos removeParrafo: elimina un parrafo de la lista de parrafos clearAllParrafos: elimina todos los elementos de la lista de parrafos getAllParrafos: devuelve un iterador de la lista de parrafos getParrafos: devuelve la lista de parrafos setParrafos: asigna al atributo una lista de parrafos CapituloStructure: es el constructor de la clase

CLASE PARRAFOSTRUCTURE	
 **Figura 93:** Diseño: Clase ParrafoStructure	Clase que representa la estructura de un párrafo de un capitulo
ATRIBUTOS	
 **Figura 94:** Diseño: Atributos ParrafoStructure	Texto: es el texto del párrafo Imagen: es la dirección de la imagen que se extrae del documento original Subparrafos: es la lista de los subpárrafos de este capitulo
MÉTODOS	
 **Figura 95:** Diseño: Métodos ParrafoStructure	ParrafoStructure: constructor de la clase setTexto: asigna el texto del párrafo al atributo setImagen: asigna la ruta de la imagen extraída del documento original al atributo addSubParrafo: añade un elemento a la lista de subpárrafos. removeSubParrafo: elimina un elemento de la lista de subpárrafos. clearAllSubParrafo: elimina toda la lista de subpárrafos. getAllSubParrafos devuelve un iterador de la lista de subpárrafos setSubParrafos asigna una lista a los subpárrafos getTexto: devuelve el texto de ese párrafo getImagen: devuelve la ruta de la imagen

CLASE SUBPARRAFOSTRUCTURE

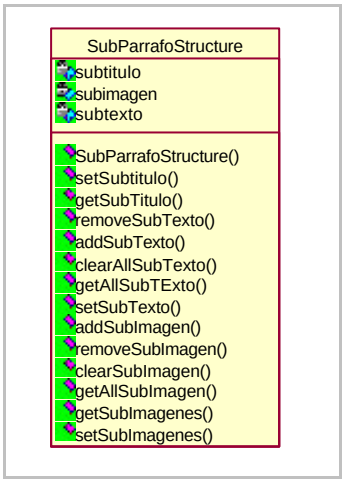


Figura 96: Diseño: Clase SubParrafoStructure

Clase que representa la estructura de un subpárrafo de un párrafo.

ATRIBUTOS

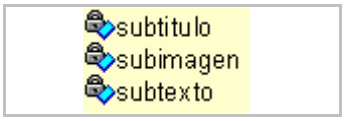


Figura 97: Diseño: Atributos SubParrafoStructure

Subtitulo : representa el titulo del subpárrafo en caso de existir

Subimagen: representa la lista de las rutas de las imágenes que se encuentran en el subpárrafo en caso de existir

Subtexto: representa una lista de elementos que es texto

MÉTODOS

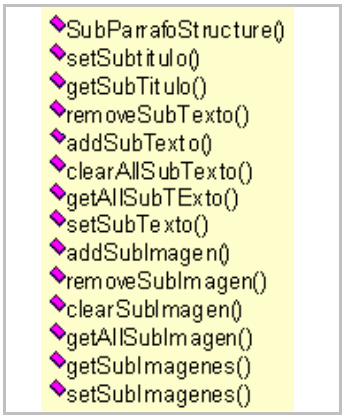


Figura 98: Diseño: Métodos

SubParrafoStructure: es el constructor de la clase

setSubTitulo: asigna el subtitulo al atributo

getSubtitulo: devuelve el subtitulo del subpárrafo.

removeSubTexto: elimina un elemento de la lista de subtexto

addSubTexto: añade un elemento a la lista de subtexto

clearAllSubTexto: elimina todos los elementos de la lista de subtexto.

setSubTexto: asigna una lista a la lista de subtextos

addSubImagenes: añade un elemento a la lista de subimágenes.

removeSubImagen: elimina un elemento de la lista de subimágenes

SubParrafoStructure	clearSubImagen: elimina todos los elementos de la lista de subImagen getSubImagenes: devuelve la lista de las subimágenes setSubImagenes: asigna una lista al atributo subImagenes

3.2.3 Nuevas funcionalidades de la arquitectura

A continuación se muestran dos nuevas funcionalidades que se añade a la primera aproximación de la arquitectura desarrollada, y las tres últimas modifican las funcionalidades iniciales de la fase de análisis:

FUNCIONALIDADES
21. Inicializar la colección
22. Modificar la colección
23. Añadir un documento original a la colección
24. Añadir un directorio de documentos originales a la colección
25. Eliminar un documento de la colección

3.2.4 Diagrama de Casos de uso general

Después de esta pequeña modificación en las funcionalidades del sistema, el diagrama de casos de uso queda como sigue:

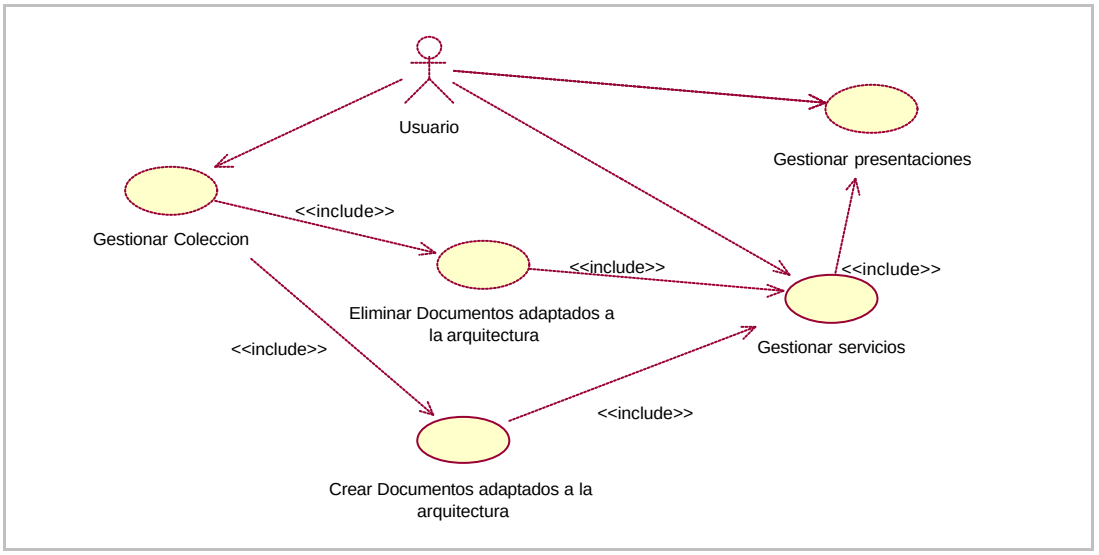


Figura 99: Diseño: Diagrama de casos de uso de la arquitectura

Se ha realizado una pequeña modificación. A partir de ahora ya no va a ser el usuario directamente el que va a crear y eliminar los documentos adaptados a la arquitectura, sino que esto se realizará a través de la colección. A continuación se procede a explicar con más detalle esos casos de uso.

3.2.5 Diagramas de casos de uso y de secuencia

A continuación se van a revisar todos los aspectos que se describieron en la fase de análisis para realizar mejoras y adaptarlo a estas nuevas necesidades del sistema. Por lo tanto se detallarán esos aspectos que no se describieron en profundidad en la etapa anterior. Es necesario resaltar el uso de un fichero de configuración en el que se incluyen los nombres de los servicios sobre documentos individuales así como todos los parámetros necesarios para el funcionamiento del sistema.

3.2.5.1 Caso de uso: Gestionar Colección

Este caso de uso forma parte de una de las novedades de la fase de diseño. Fue necesario incorporarlo ya que debido al uso de un fichero de configuración, ciertas tareas se iban a ver realizadas de una forma automática sin necesidad de un usuario que las instanciara.

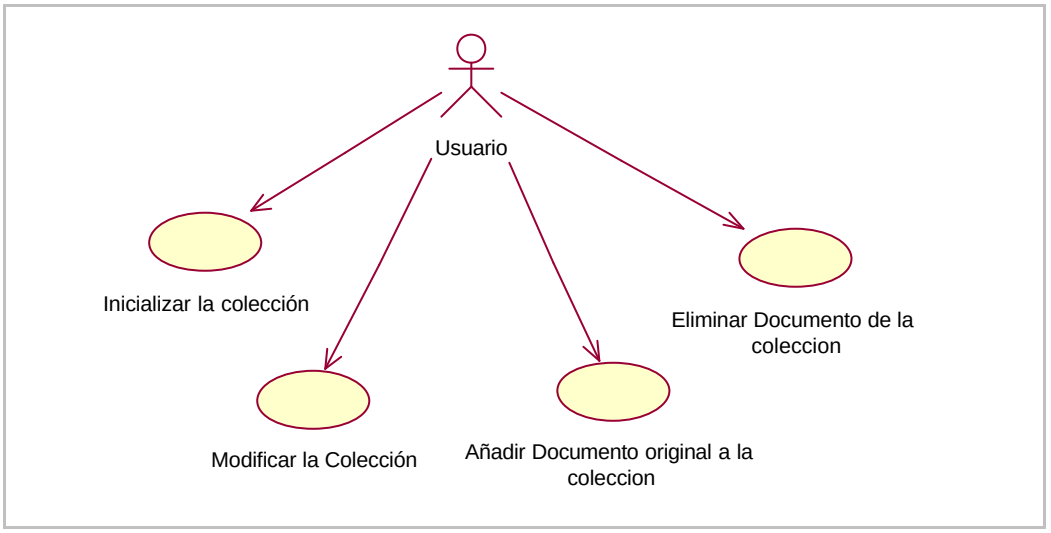


Figura 100: Diseño: Diagrama de caso de uso: Gestionar Colección

OBJETIVOS
Este caso de uso es el responsable de que el usuario gestiona la colección, es decir, desde este punto se podrá inicializar la colección para de ese modo en el caso de haber modificado el fichero de configuración, el sistema sea capaz comprobar si todas las visiones y servicios se encuentran disponibles. Lo mismo

ocurre con "*Modificar Colección*". Se ha incluido el añadir y eliminar documento a la colección para facilitar el uso de la arquitectura por parte del usuario final. De este modo todo lo que se realice internamente dentro de la colección será completamente transparente.
ACTORES
Usuario
PRECONDICIONES
El fichero de configuración debe de estar correctamente configurado
POSTCONDICIONES
No existen.
FUNCIONES QUE CUMPLE
22, 23, 24, 25

Escenario: Añadir Documento original a la colección

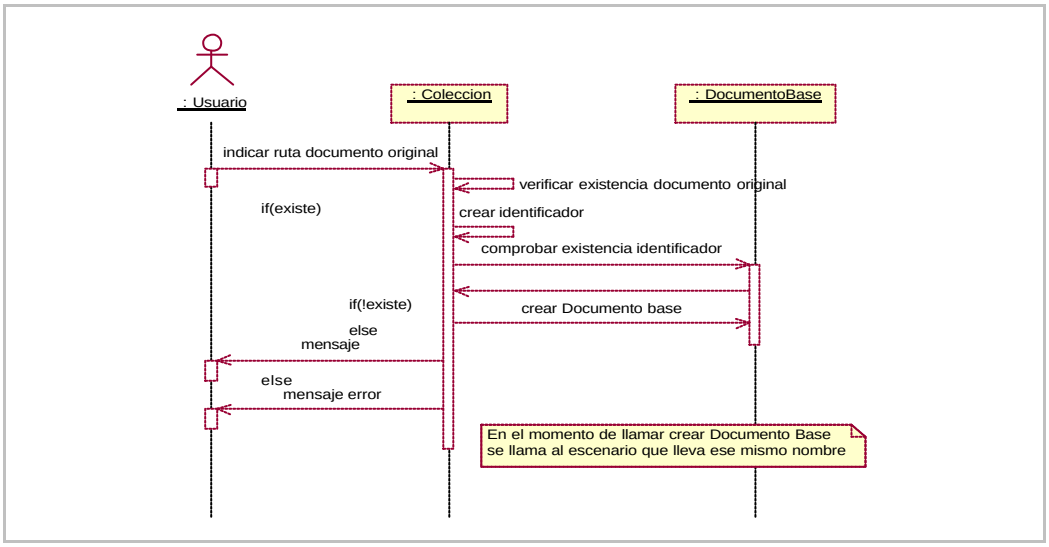


Figura 101: Diseño: Diagrama de secuencia: Añadir Documento original a la colección

El diagrama de colaboración de este escenario:

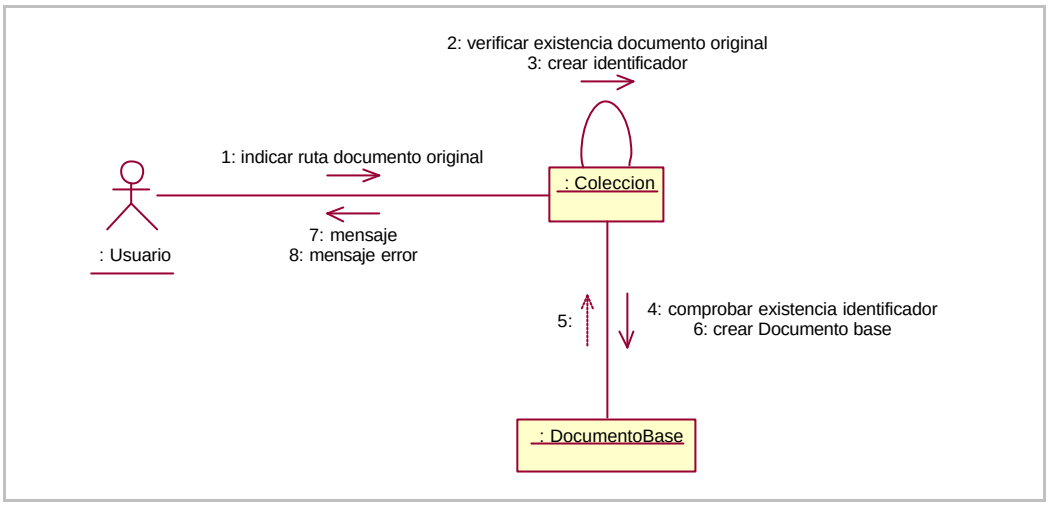


Figura 102: Diseño: Diagrama de colaboración: Añadir Documento original a la colección

Se puede observar tanto en el diagrama de secuencia como en el diagrama de colaboración que *"Añadir documento original a la colección"* es el responsable de instanciar "*Crear Documento base*".

Escenario: Eliminar Documento de la colección

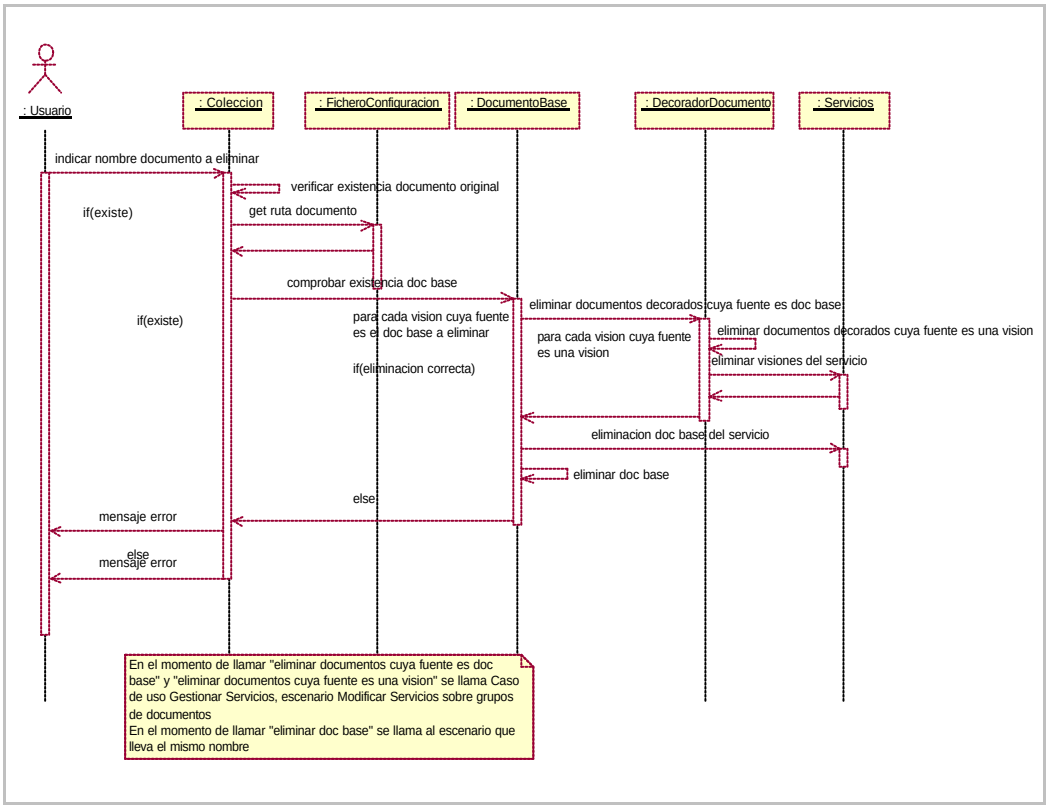


Figura 103: Diseño: Diagrama de secuencia: Eliminar Documento de la colección

El diagrama de colaboración de este escenario:

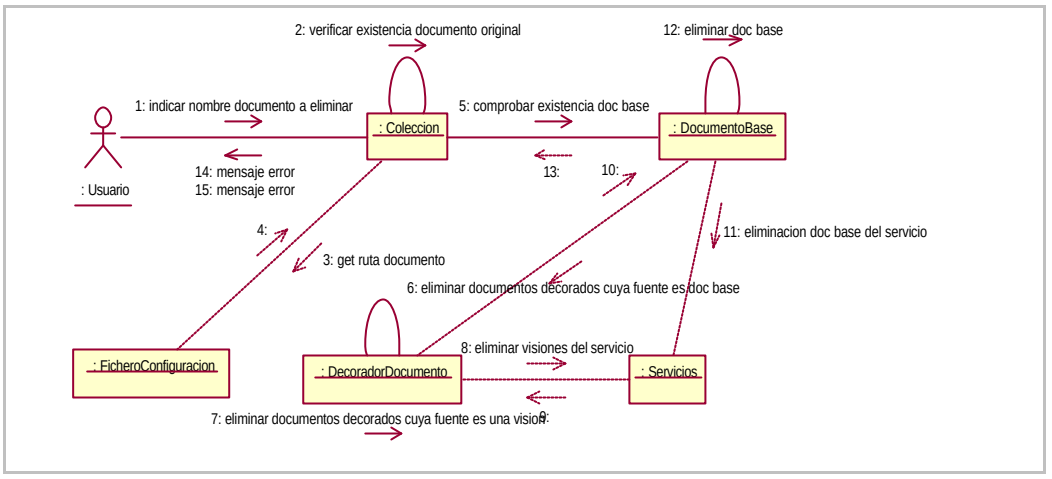


Figura 104: Diseño: Diagrama de colaboración: Eliminar Documento de la colección

Se puede observar tanto en el diagrama de secuencia como en el diagrama de colaboración que "*Eliminar documento de la colección*" es el responsable de instanciar "*Eliminar Documento base*".

Escenario: Inicializar la colección

Este escenario tiene lugar nada más arrancar la herramienta, en él se comprueba que existan todas las visiones asociadas a todos los documentos base y que estén agregados a los servicios correspondientes.

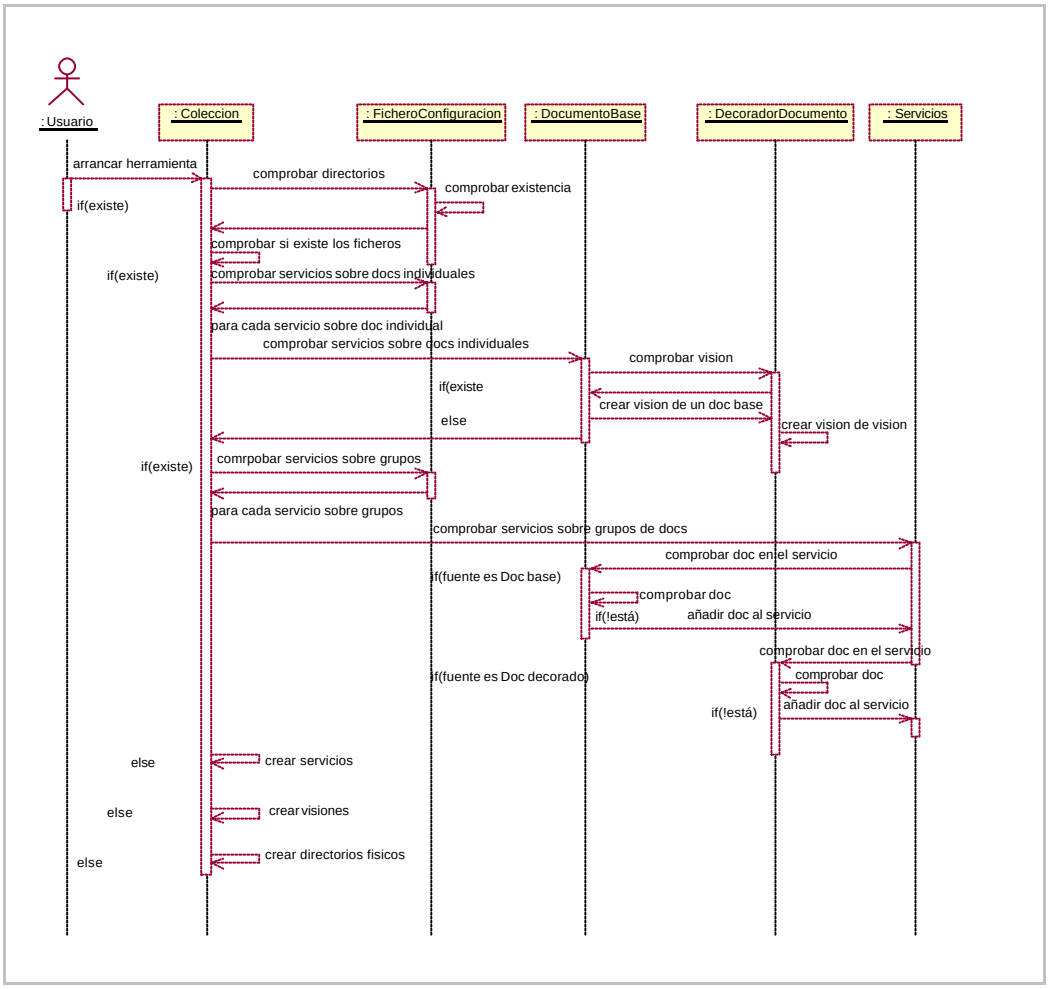


Figura 105: Diseño: Diagrama de secuencia: Inicializar la colección

El diagrama de colaboración de este escenario:

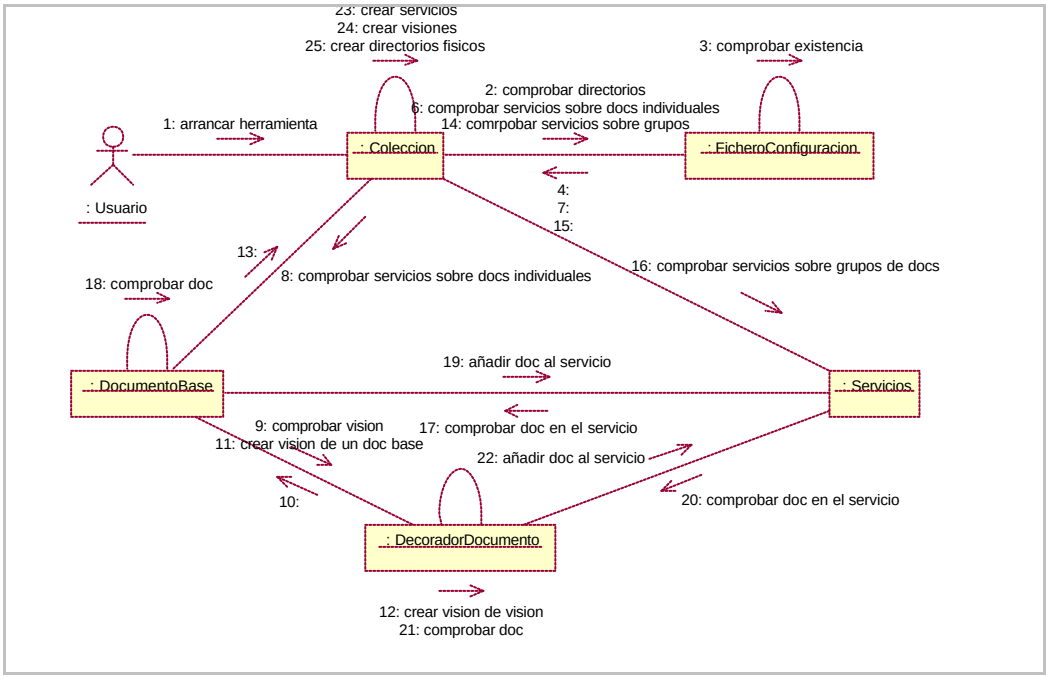


Figura 106: Diseño: Diagrama de colaboración: Inicializar la colección

Se puede observar tanto en el diagrama de secuencia como en el diagrama de colaboración que "*Crear una visión de un documento base*" y "*crear una visión de una visión*" no forman parte de este caso de uso pero son instanciados desde éste. Lo mismo ocurre con "*Añadir un documento al servicio*".

Escenario: Modificar la colección

Este escenario soporta el hecho de que en el caso de modificar el fichero de configuración en tiempo de ejecución, esas modificaciones se trasladen a modificar la colección en respuesta a función de los cambios realizados.

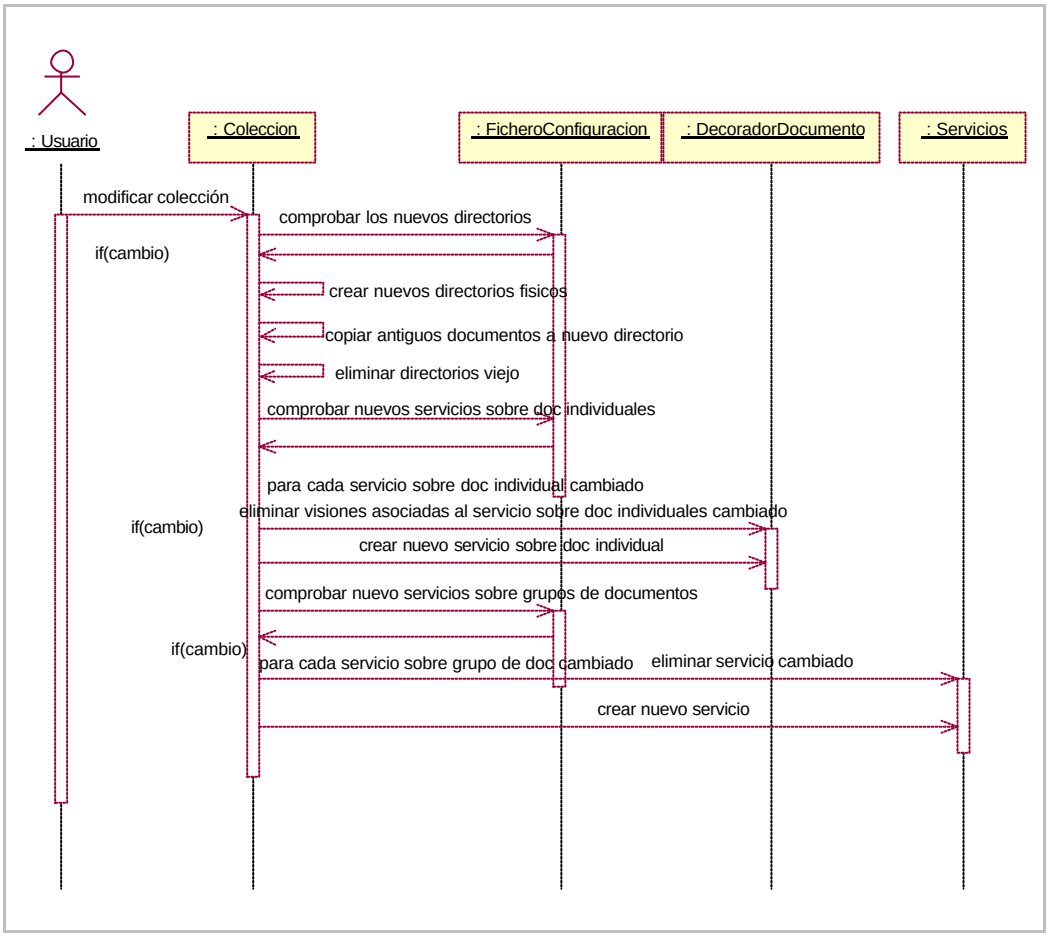


Figura 107: Diseño: Diagrama de secuencia: Modificar la colección

El diagrama de colaboración de este escenario:

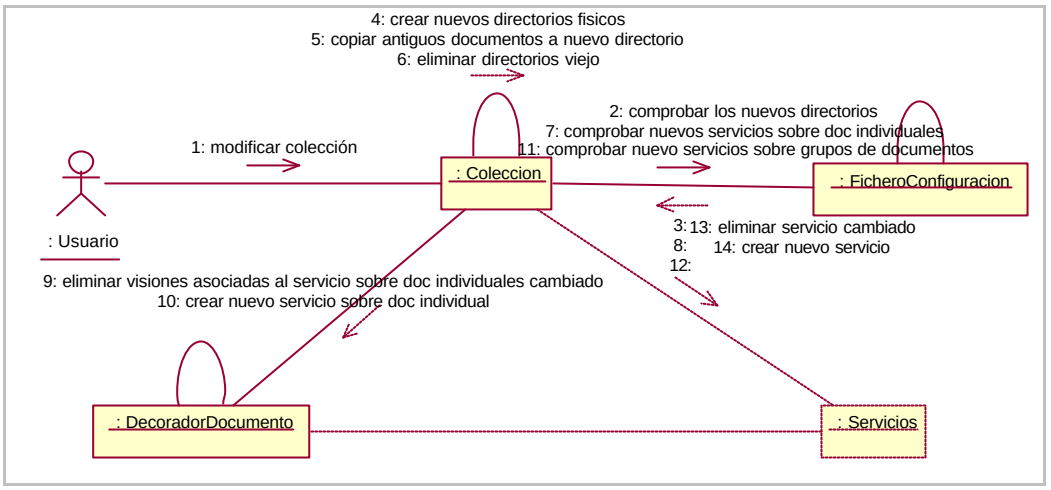


Figura 108: Diseño: Diagrama de colaboración: Modificar la colección

Se puede observar tanto en el diagrama de secuencia como en el diagrama de colaboración que "*Crear Nuevo servicio sobre documentos individuales*" hace alusión a "*Crear visión de un documento base*" y a "*Crear una visión de una visión*". Estos escenarios son instanciados desde éste. Lo mismo ocurre con "*Eliminar servicios sobre documentos individuales"* que hace alusión a "*Eliminar una visión de un documento base*" y a "*Eliminar una visión de una visión*".

3.2.5.2 Caso de uso: Crear Documentos adaptados a la arquitectura

Este diagrama de casos de uso se mantiene prácticamente igual que el descrito en la etapa de análisis, la diferencia única con el anterior es que ya no es el usuario el que pasa los parámetros directamente a la arquitectura, sino que se va a hacer mediante un fichero de configuración. Por tanto la precondición va a ser que ese fichero tiene que estar correctamente configurado.

Otro aspecto que ha cambiado respecto a la etapa de análisis donde a la hora de añadir un documento a la colección este escenario no tenía relevancia. Sin embargo, según el planteamiento seguido en el diseño es necesario incluirlo ya que en esta etapa se incorporó a "*Gestionar Colección*".

En los diagramas de secuencia y colaboración que aquí se van a describir, es preciso matizar que quien realiza la llamada va a ser el caso de uso "*Gestionar Colección*" y que así se ha hecho en los diagramas, es decir, se parte de la llamada que realiza ese caso de uso para de ese modo conseguir un mayor entendimiento por parte del lector.

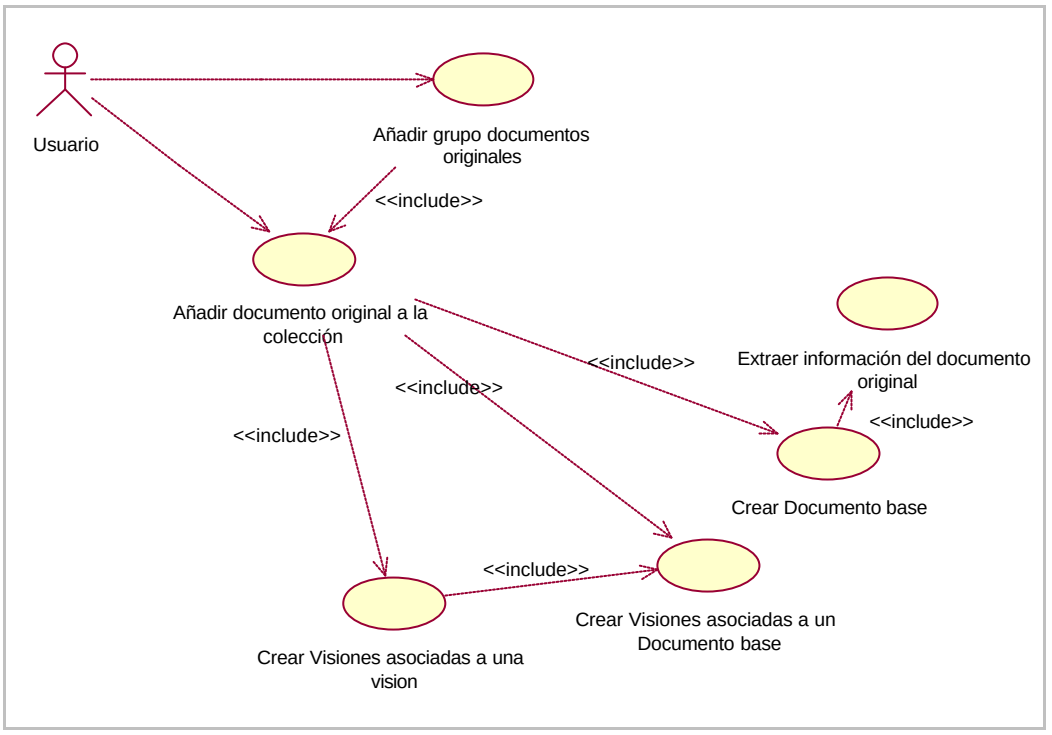


Figura 109: Diseño: Diagrama de caso de uso: Crear Documento adaptado a la arquitectura

Escenario: Crear Documento Base

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

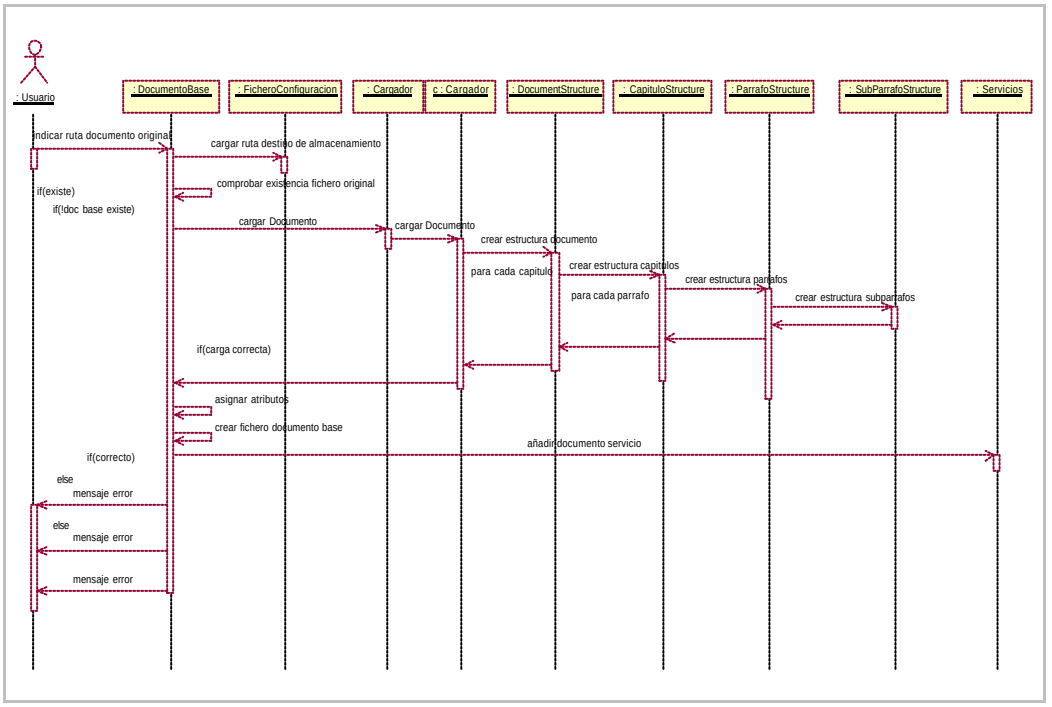


Figura 110: Diseño: Diagrama de secuencia: Crear Documento Base

El diagrama de colaboración de este escenario:

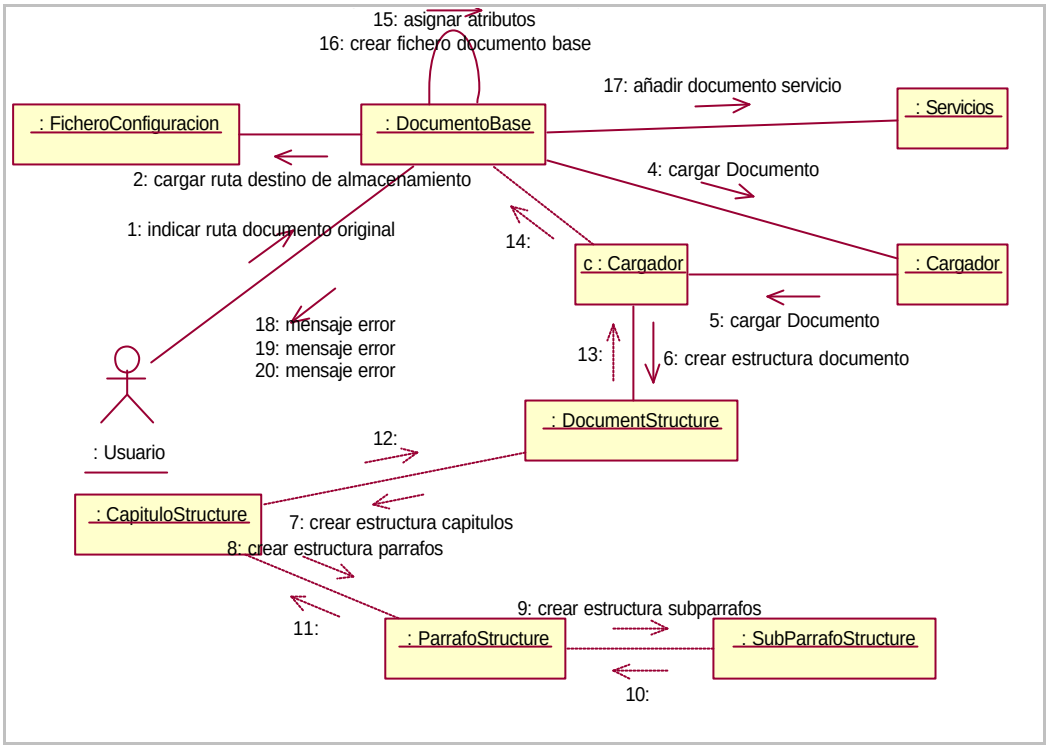


Figura 111: Diseño: Diagrama de colaboración: Crear Documento Base

Escenario: Crear Visiones asociadas a un documento Base

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

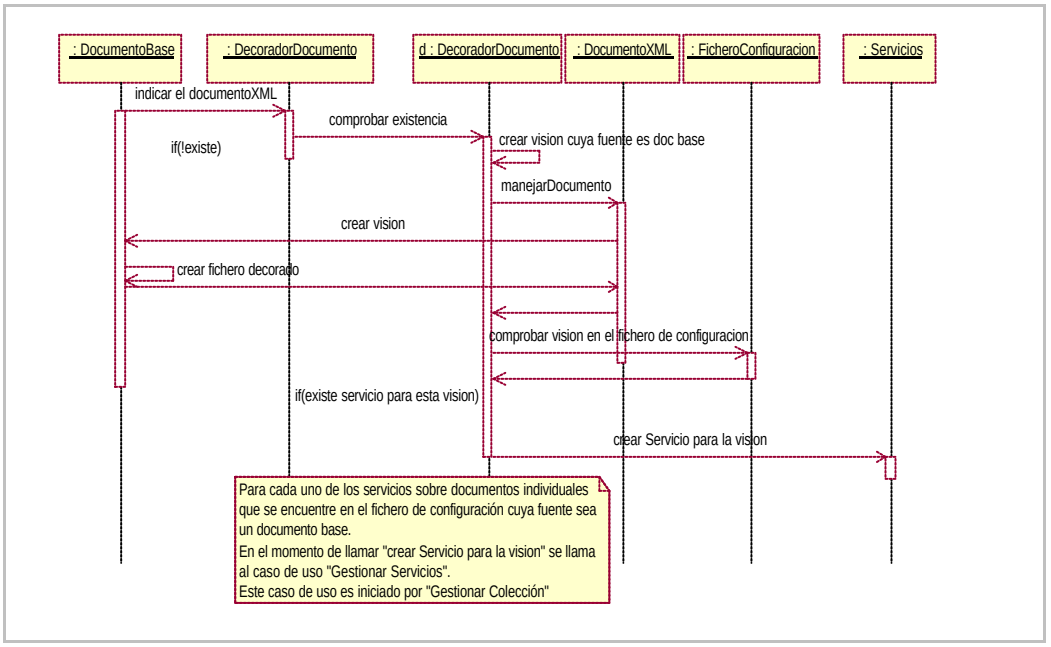


Figura 112: Diseño: Diagrama de secuencia: Crear Visiones asociadas a un documento Base

El diagrama de colaboración de este escenario:

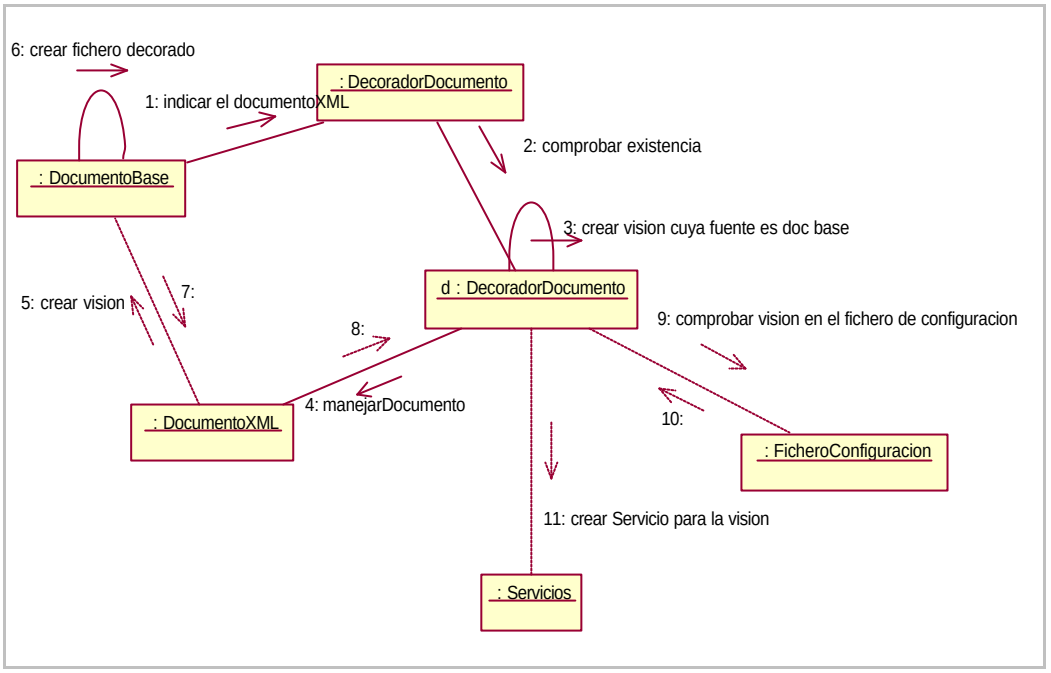


Figura 113: Diseño: Diagrama de colaboración: Crear Visiones asociadas a un documento Base

Escenario: Crear Visiones asociadas a visiones

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

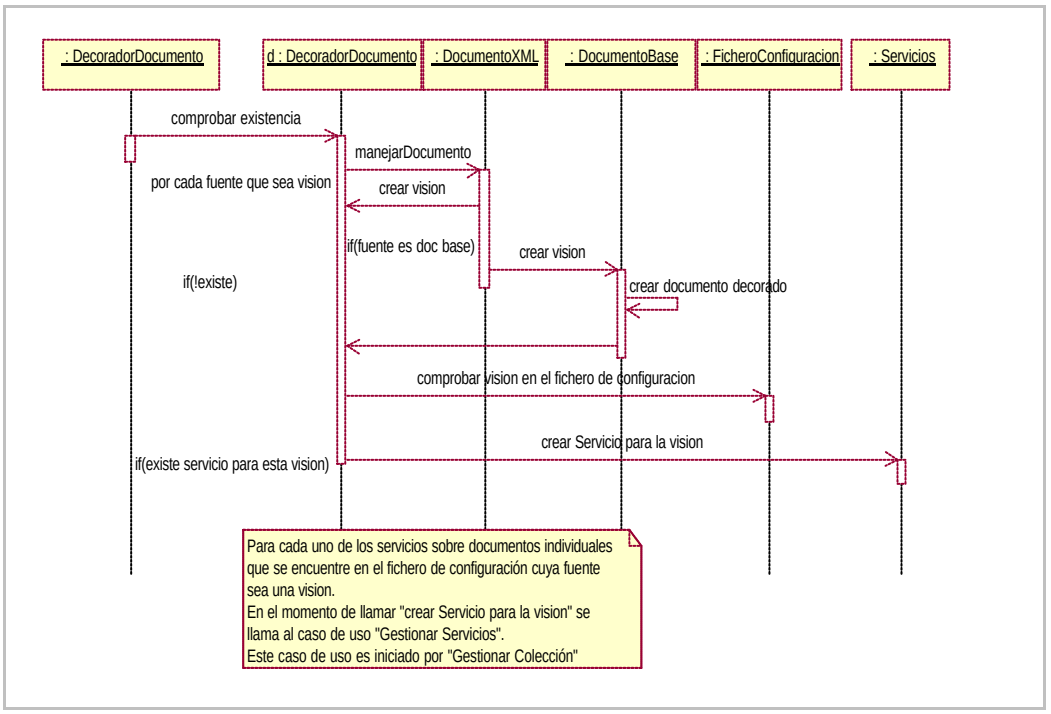


Figura 114: Diseño: Diagrama de secuencia: Crear Visiones asociadas a visiones

El diagrama de colaboración de este escenario:

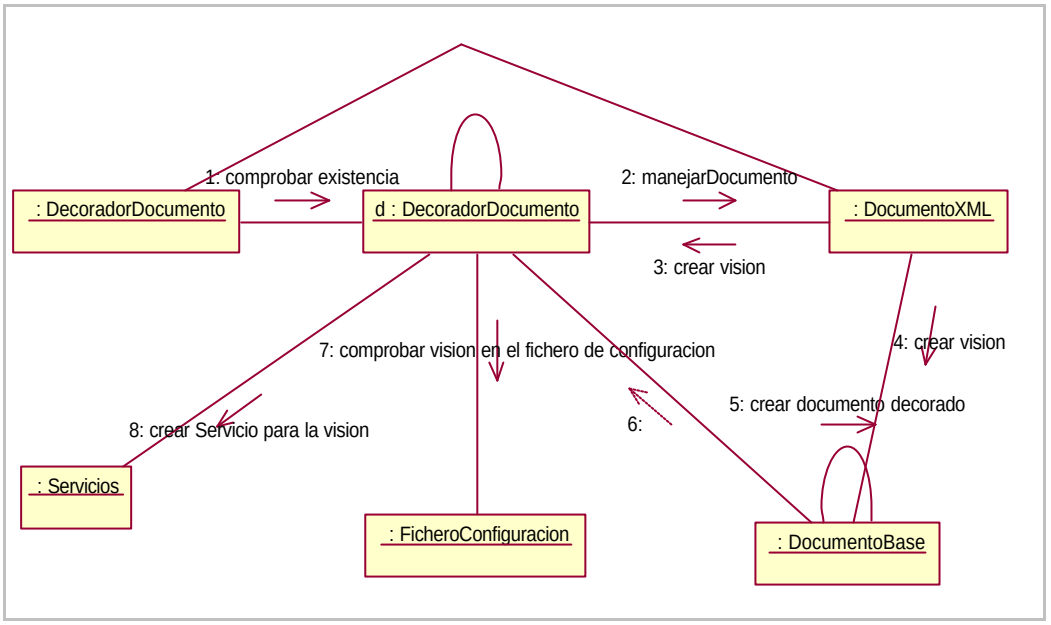


Figura 115: Diseño: Diagrama de colaboración: Crear Visiones asociadas visiones

3.2.5.3 Caso de uso: Eliminar Documento adaptado a la arquitectura

Este diagrama de casos de uso se mantiene igual que el de la etapa del análisis, aunque con la diferencia de que ya no es el usuario el que le pasa los parámetros directamente a la arquitectura, sino que se va a hacer mediante un fichero de configuración. Por tanto la precondición va a ser que ese fichero tiene que estar correctamente configurado.

En los diagramas de secuencia y colaboración que aquí se van a describir, es preciso matizar que quien realiza la llamada va a ser el caso de uso "*Gestionar Colección*" y que así se ha hecho en los diagramas, es decir, se parte de la llamada que realiza ese caso de uso para de ese modo conseguir un mayor entendimiento por parte del lector.

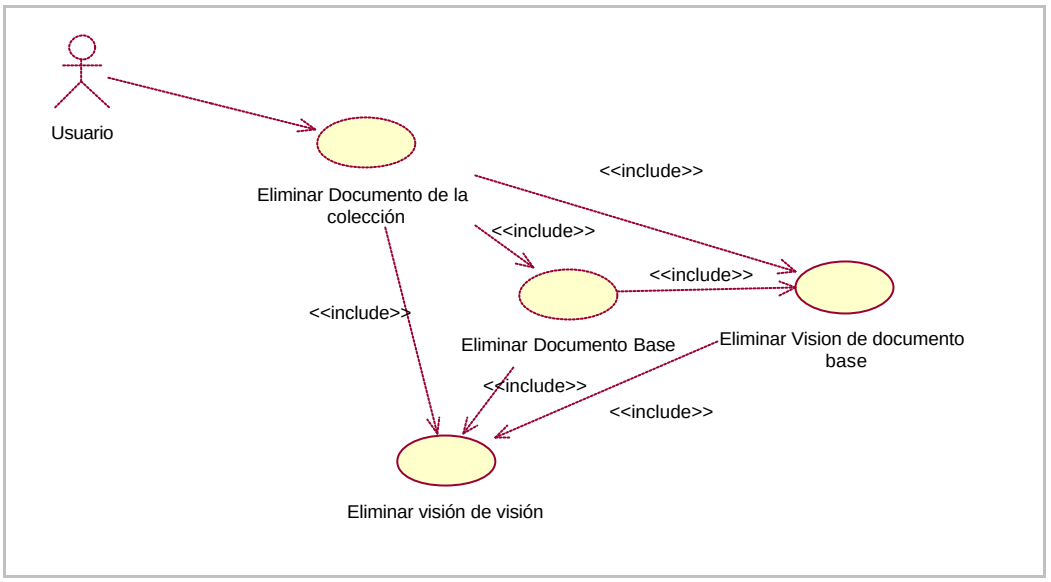


Figura 116: Diseño: Caso de uso: Eliminar Documento adaptado a la arquitectura

Escenario: Eliminar Documento Base

Este escenario sigue la misma filosofía que en la fase de análisis, es decir, cuando se elimina un documento base, se eliminan todas las visiones asociadas ya bien sea las que provienen de una fuente que es un documento base, ya bien sea las que provienen de una fuente que es una visión. El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

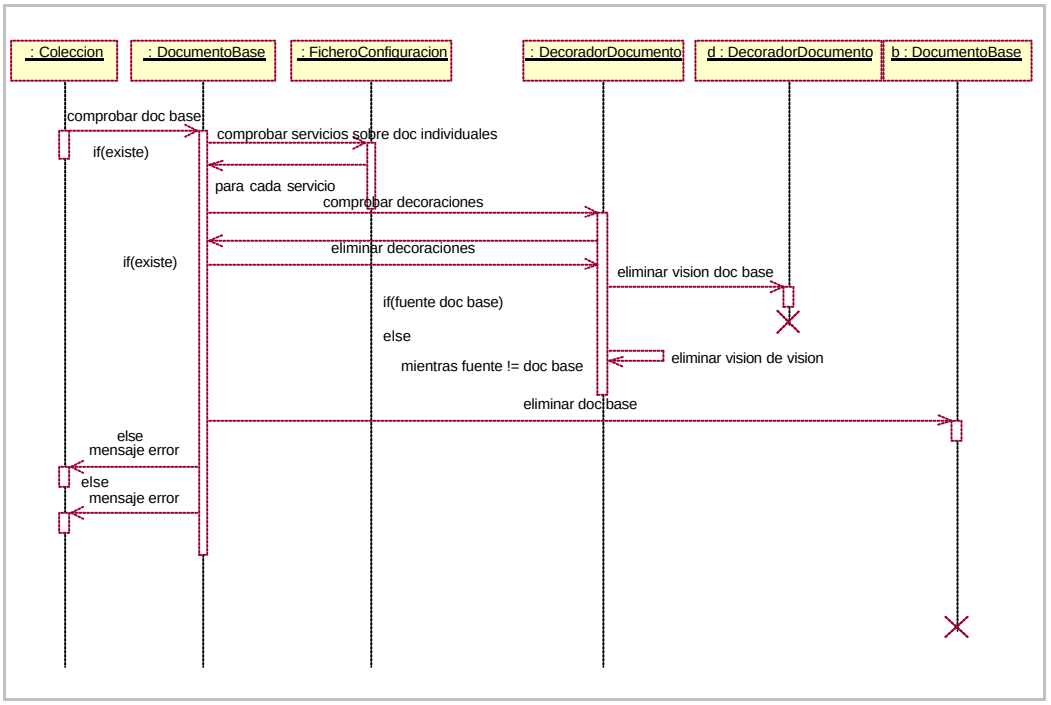


Figura 117: Diseño: Diagrama de secuencia: Eliminar Documento Base

El diagrama de colaboración de este escenario:

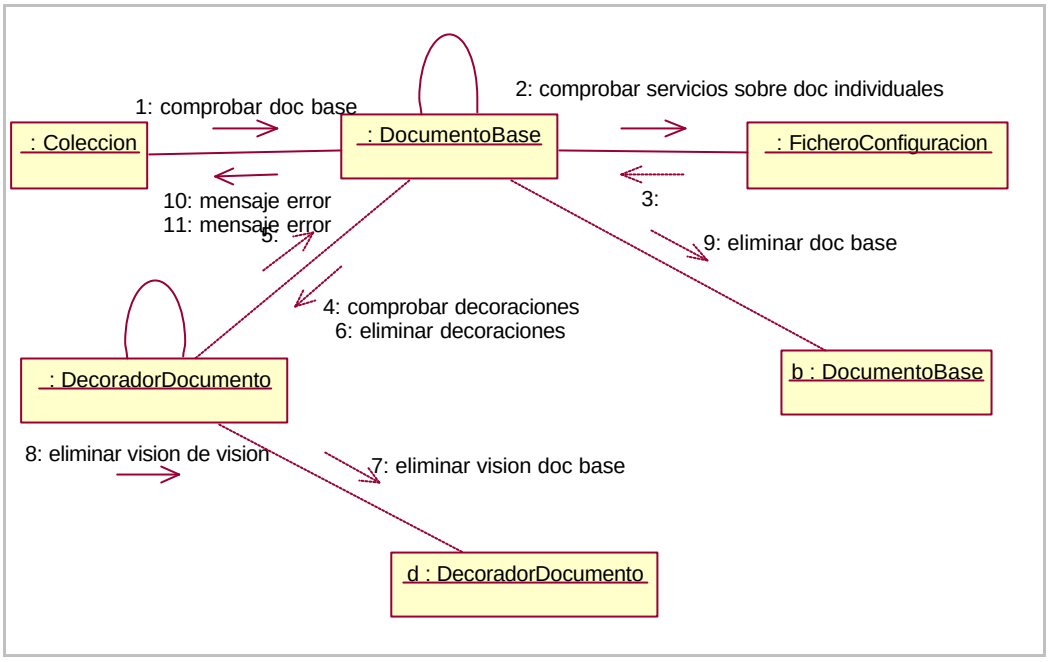


Figura 118: Diseño: Diagrama de colaboración: Eliminar Documento Base

Se puede observar como este escenario realiza la llamada a "*Eliminar visión de un documento base*"y a "*Eliminar una visión de visión*".

Escenario: Eliminar visión de documento base

Este escenario puede ser instanciado de dos maneras diferentes.

Por un lado puede ocurrir cuando se elimina un documento base. Pero por otro también podría ocurrir en el momento de modificar la colección. Por lo tanto se van a considerar las dos situaciones. Se comenzará por el caso de que se esté eliminando un documento base.

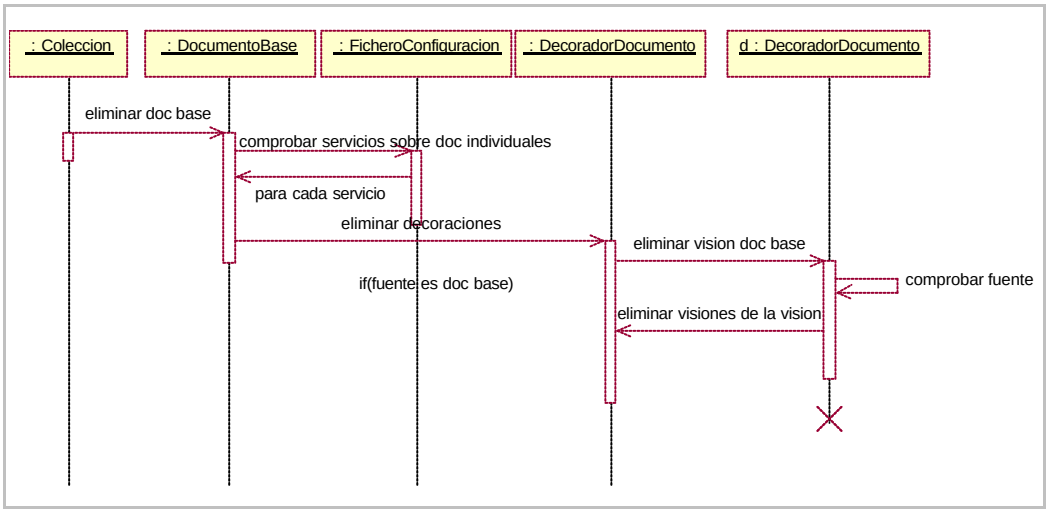


Figura 119: Diseño: Diagrama de secuencia: Eliminar visión de documento base1

El diagrama de colaboración de este escenario:

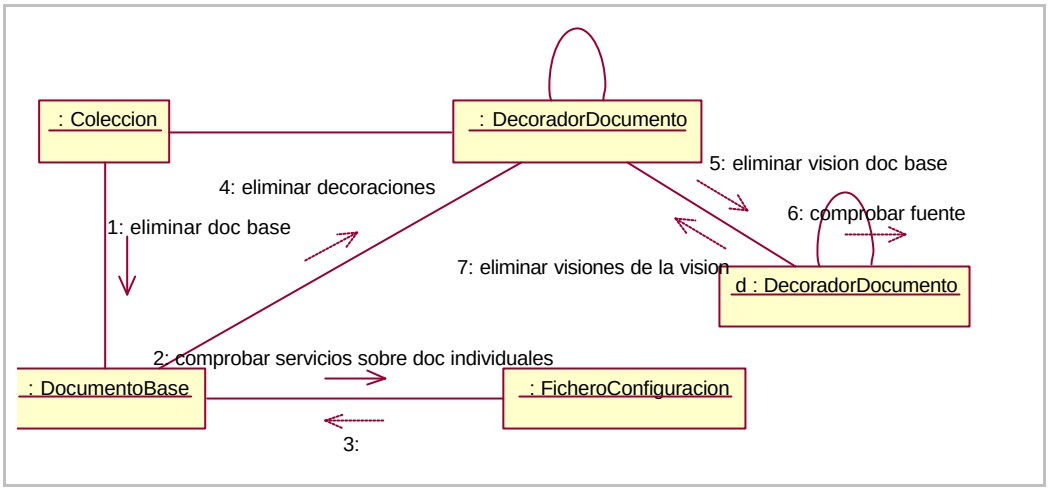


Figura 120: Diseño: Diagrama de colaboración: Eliminar visión de documento base1

En el caso de que se esté modificando la colección los diagramas son los siguientes:

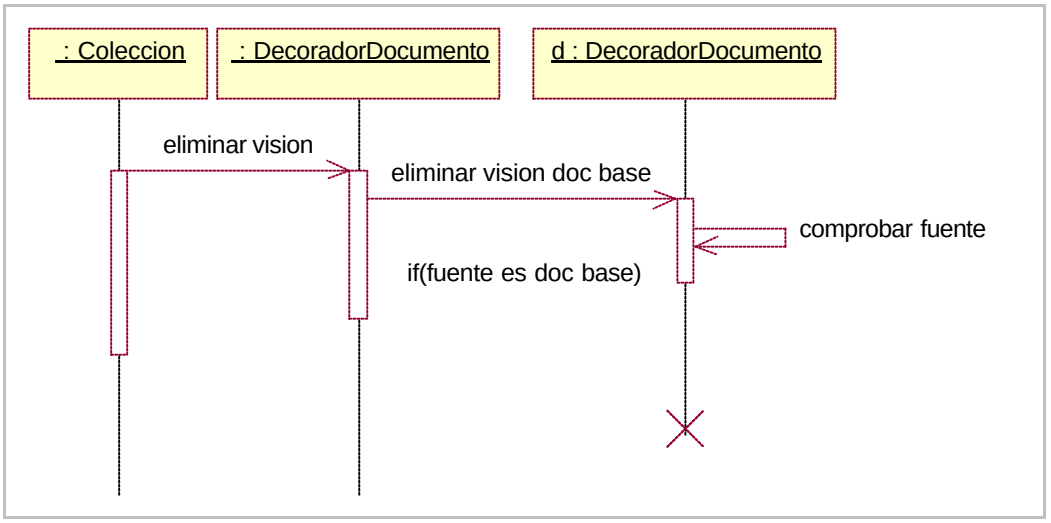


Figura 121: Diseño: Diagrama de secuencia: Eliminar visión de documento base2

El diagrama de colaboración de este escenario:

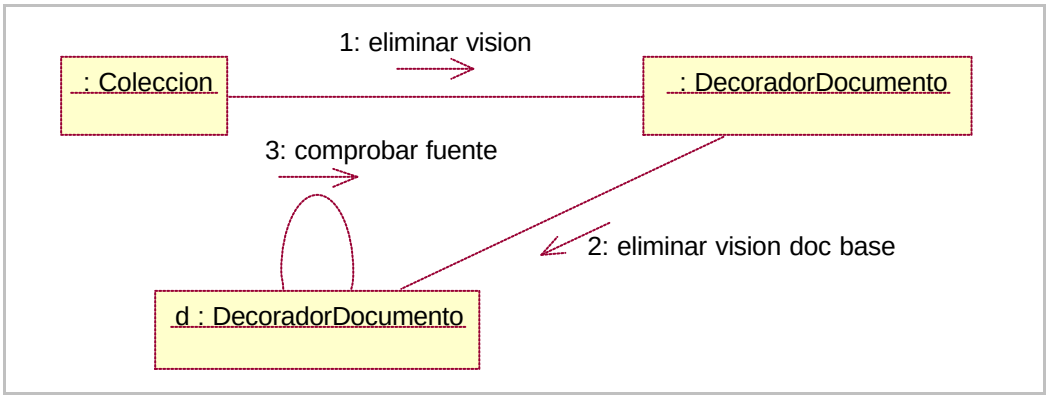


Figura 122: Diseño: Diagrama de colaboración: Eliminar visión de documento base2

Escenario: Eliminar visión de visión

Este escenario puede ser instanciado de dos maneras diferentes.

Por un lado puede ocurrir cuando se elimina un documento base. Éste a su vez elimina las visiones del documento base y elimina también las visiones de visiones. Pero podría ocurrir en el momento de modificar la colección. Por lo tanto, se van a considerar ambas situaciones. Se comenzará por el caso de que se esté eliminando un documento base y éste a su vez elimine las visiones de un documento base.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

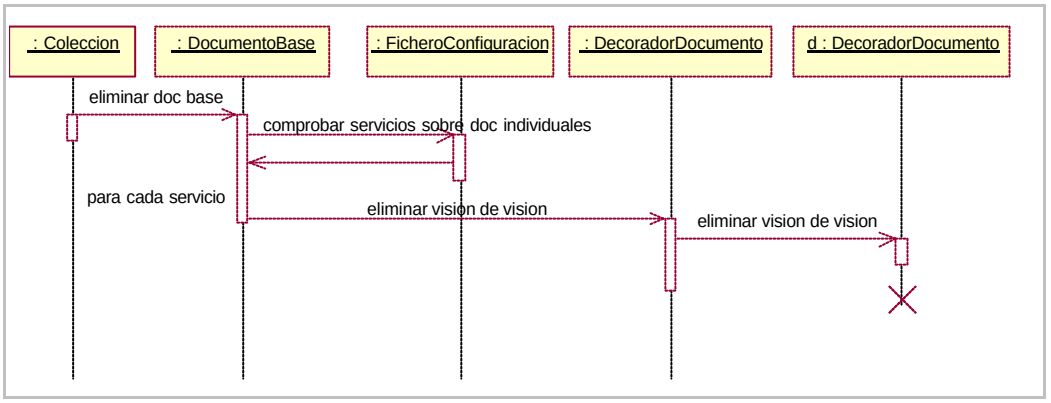


Figura 123: Diseño: Diagrama de secuencia: Eliminar visión de visión1

El diagrama de colaboración de este escenario:

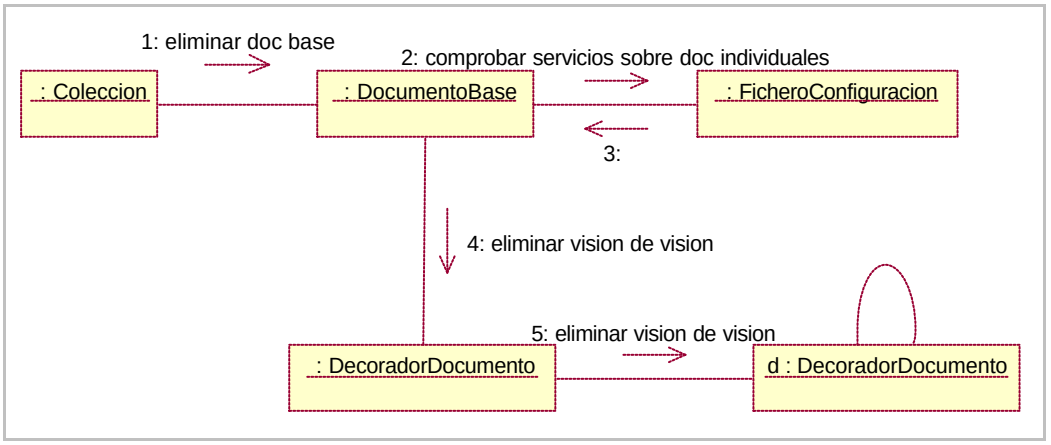


Figura 124: Diseño: Diagrama de colaboración: Eliminar visión de visión1

En el caso de que se esté modificando la colección los diagramas son los siguientes:

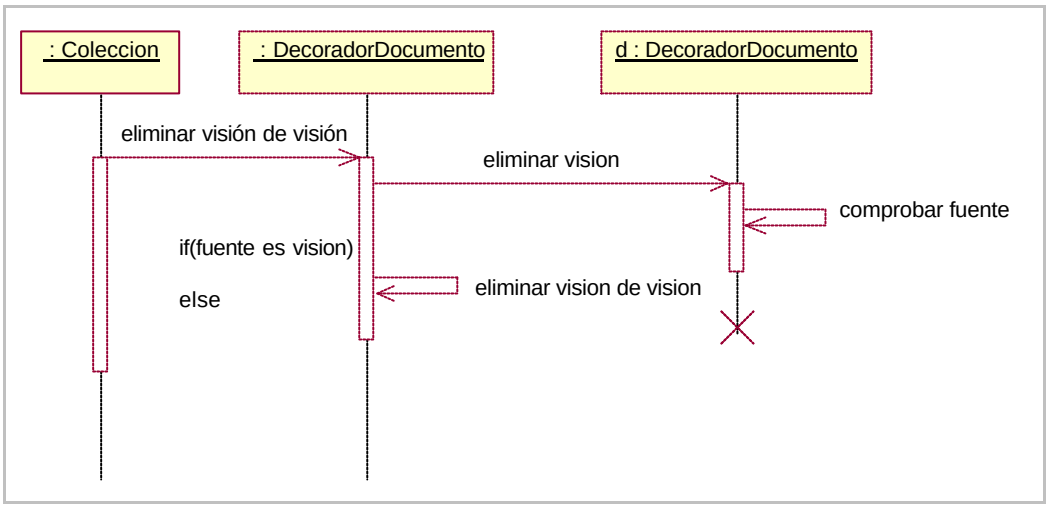


Figura 125: Diseño: Diagrama de secuencia: Eliminar visión de visión2

El diagrama de colaboración de este escenario:

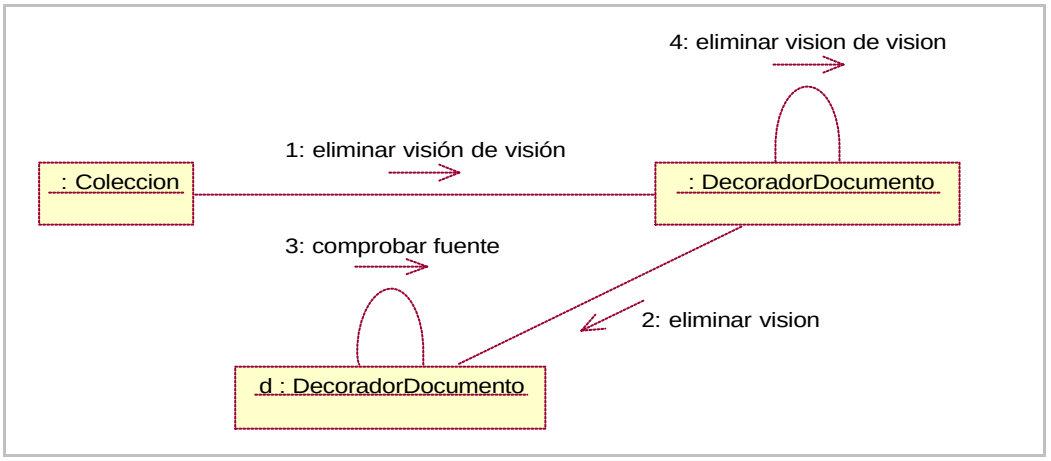


Figura 126: Diseño: Diagrama de colaboración: Eliminar visión de visión2

3.2.5.4 Caso de uso: Gestionar Servicios

Este diagrama de casos de uso se mantiene igual que el de la etapa del análisis, la única diferencia con el anterior es que ya no es el usuario el que inicia los escenarios "*Modificar servicios sobre grupo de documentos*", "*Eliminar servicios sobre grupo de documentos*" y "*Crear servicio sobre grupo de documentos*", sino que se va a realizar mediante el caso de uso "*Gestionar Colección*" que lo realizará mediante un fichero de configuración. Por tanto la precondición va a ser que ese fichero tiene que estar correctamente configurado.

En los diagramas de secuencia y colaboración que aquí se van a describir, es preciso matizar que quien realiza la llamada va a ser el caso de uso "*Gestionar Colección*" y que así se ha hecho en los diagramas, es decir, se parte de la llamada que realiza ese caso de uso para de ese modo conseguir un mayor entendimiento por parte del lector.

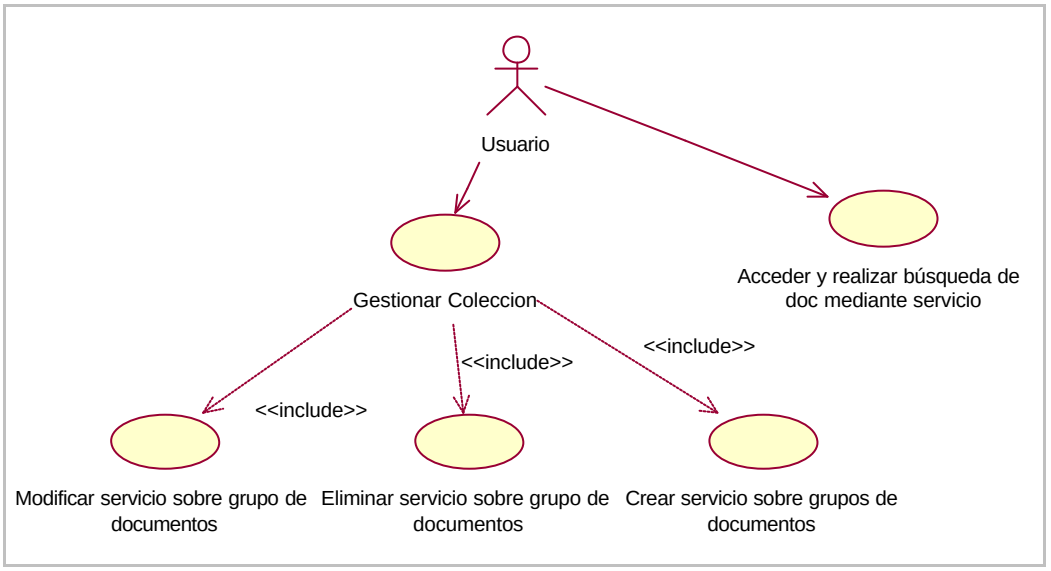


Figura 127: Diseño: Caso de uso: Gestionar Servicios

Escenario: Crear Servicio sobre grupo de documentos

Este escenario, como algunos de los vistos anteriormente, puede ser instanciado de dos maneras diferentes.

Por un lado puede ocurrir cuando se inicializa la colección pero también cuando se modifica. En el caso de "*Inicializar la colección*" se deberá de crear el servicio, pero en el caso de "*Modificar la colección*", el proceso es bastante más complejo. Así, en las situaciones donde ciertos parámetros del servicio hayan cambiado, se procederá a su eliminación para posteriormente volver a crearlo. Lo mismo ocurre en el caso de que la fuente sobre la que el servicio trabaje haya cambiado, eliminándose el servicio para luego volver a crearlo. Se ha decidido realizarlo de este modo por cuestiones de

eficiencia del sistema, ya que era bastante costoso ir incorporando documento a documento en el servicio e ir modificando. Por lo tanto, se van a considerar los dos escenarios. Se comenzará por el caso de que se esté inicializando la colección.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

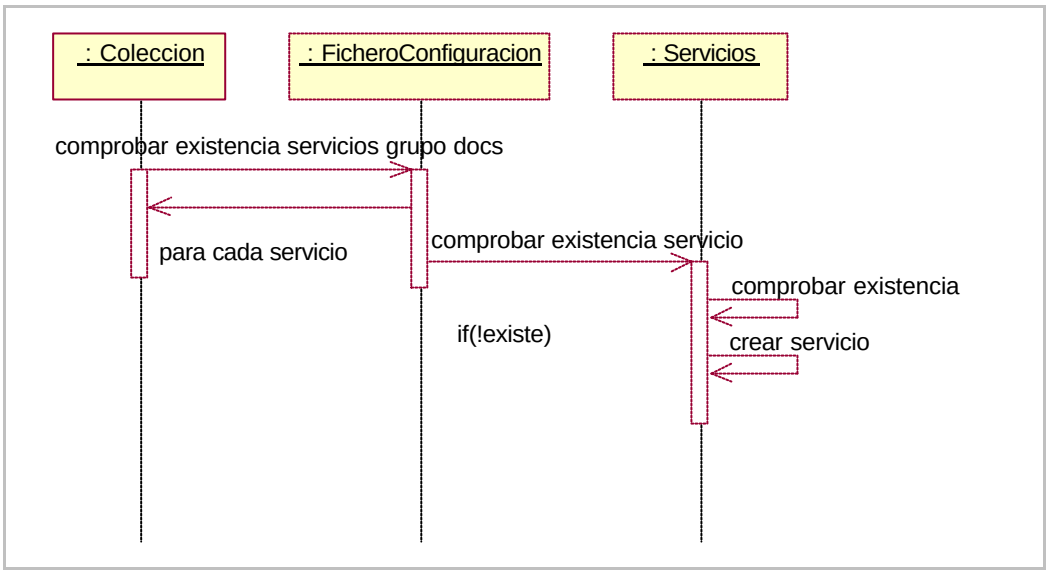


Figura 128: Diseño: Diagrama de secuencia: Crear servicio sobre grupo de documentos2

El diagrama de colaboración de este escenario:

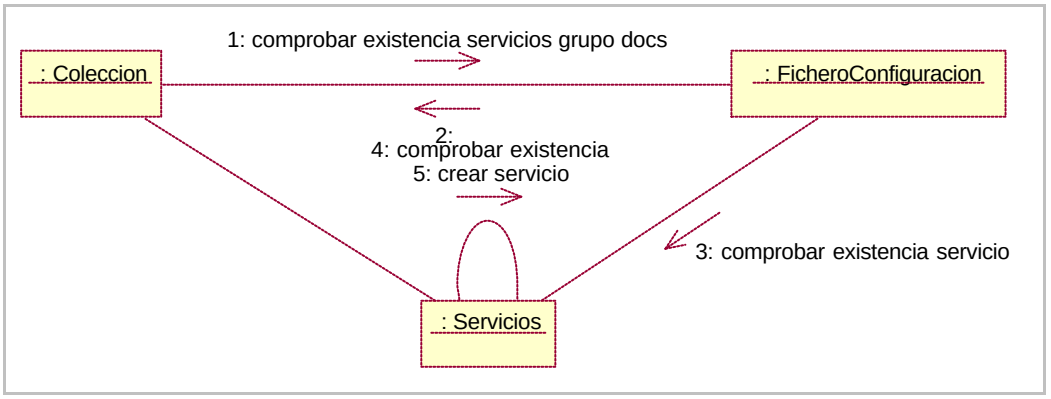


Figura 129: Diseño: Diagrama de colaboración: Crear servicio sobre grupo de documentos1

En el caso de que se esté modificando la colección los diagramas son los siguientes:

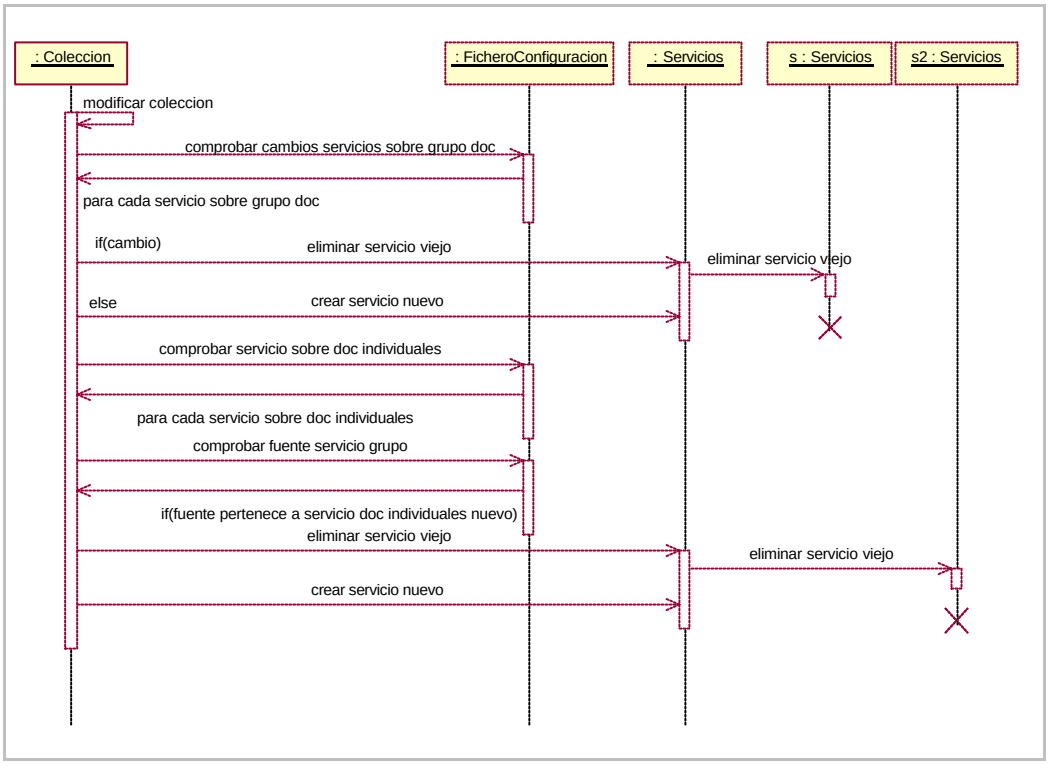


Figura 130: Diseño: Diagrama de secuencia: Crear servicio sobre grupo de documentos2

El diagrama de colaboración de este escenario:

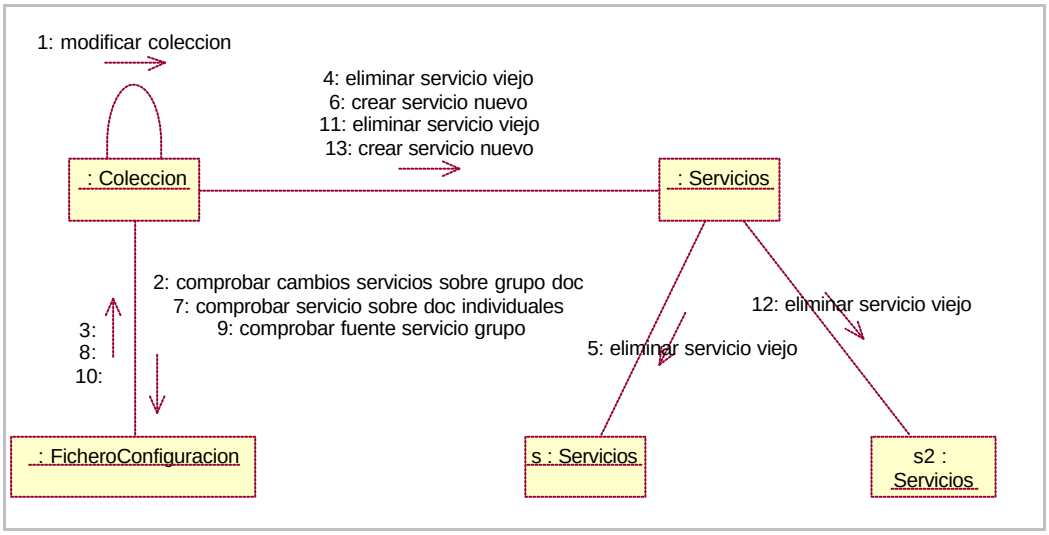


Figura 131: Diseño: Diagrama de colaboración: Crear servicio sobre grupo de documentos2

Escenario: Eliminar Servicio sobre grupo de documentos

Este escenario tiene lugar cuando se modifica la colección. Si el servicio existía pero ciertos parámetros han cambiado, se eliminará. Lo mismo ocurre si la fuente sobre el que trabaja este servicio ha cambiado, ya que sería muy costoso hacer la verificación documento a documento e ir efectuando las modificaciones.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

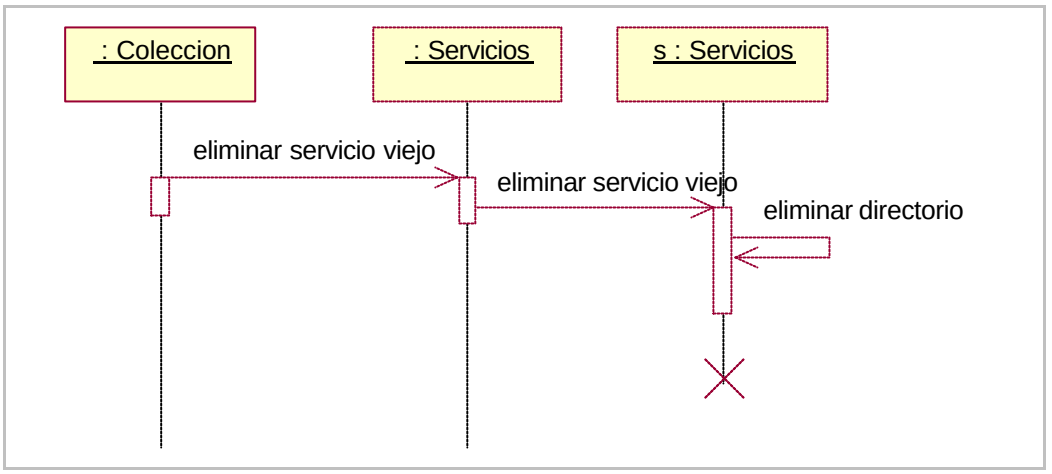


Figura 132: Diseño: Diagrama de secuencia: Eliminar servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

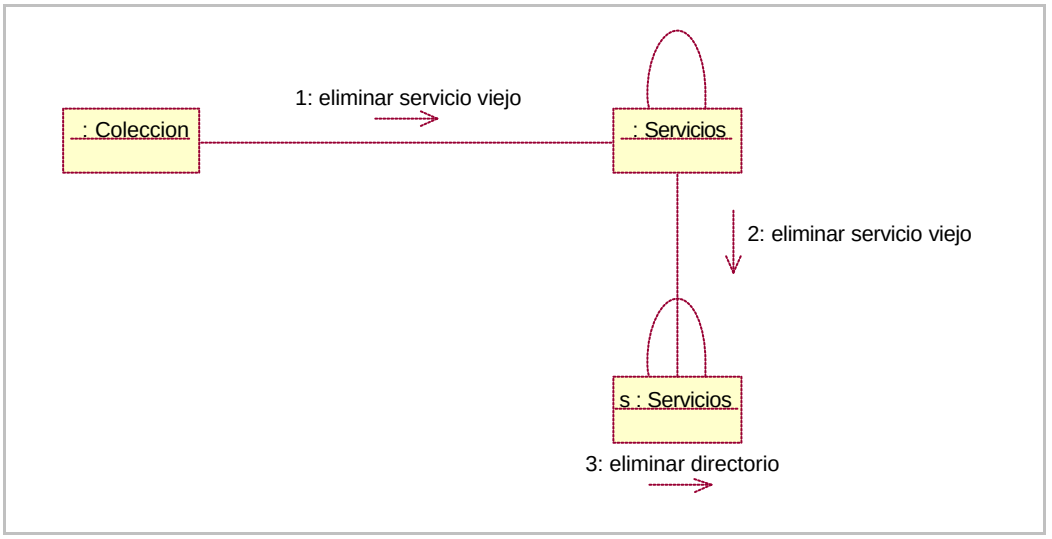


Figura 133: Diseño: Diagrama de colaboración: Eliminar servicio sobre grupo de documentos

Escenario: Modificar Servicio sobre grupo de documentos

Este escenario solo tiene lugar cuando se procede a añadir nuevos documento a la colección o cuando se procede a eliminarlos, por lo que se sigue pudiendo descomponer en dos. Por un lado "*Añadir un documento a un servicio sobre grupo de documentos*" y por el otro, "*Eliminar un documento de un servicio sobre grupos de documentos*". Se ven a continuación de una forma más detallada.

Escenario: Añadir un documento a un servicio sobre grupo de documentos

Este escenario ocurre en el momento de crear el documento, ya bien sea un documento base, ya bien sea un documento decorado, que será la fuente del servicio al que queremos añadir dicho documento.

Para mostrar esa generalidad con respecto a los documentos, no se realizará la distinción entre *DocumentoBase* y *DecoradorDocumento* sino que será un *DocumentoXML*.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

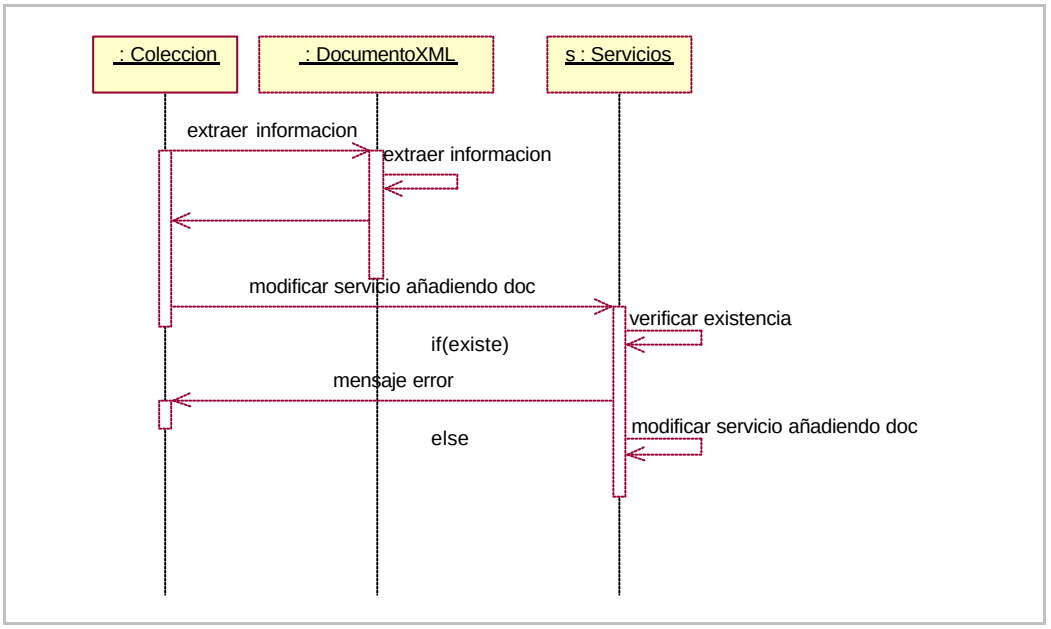


Figura 134: Análisis: Diagrama de secuencia: Añadir un documento a un servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

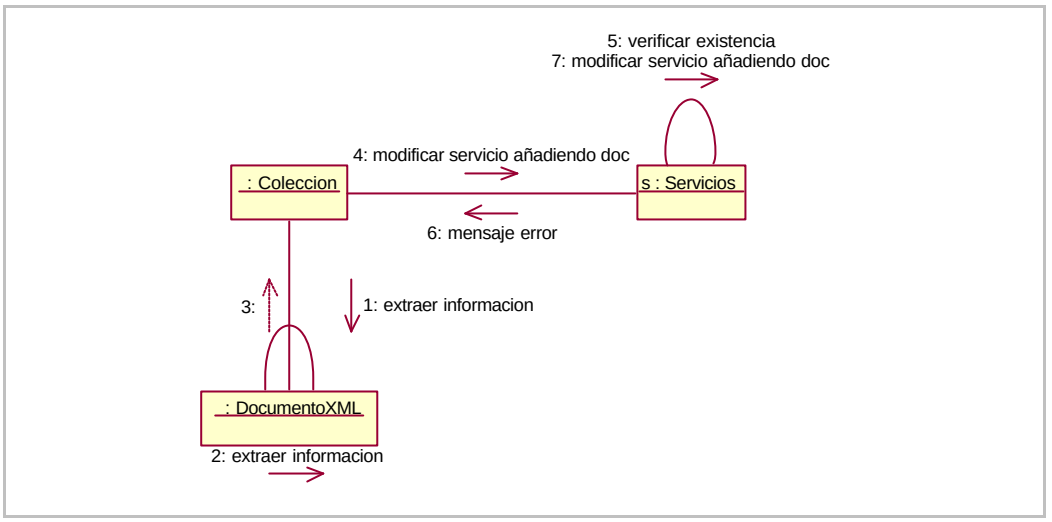


Figura 135: Diseño: Diagrama de colaboración: Añadir un documento a un servicio sobre grupo de documentos

Escenario: Eliminar un documento de un servicio sobre grupo de documentos

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

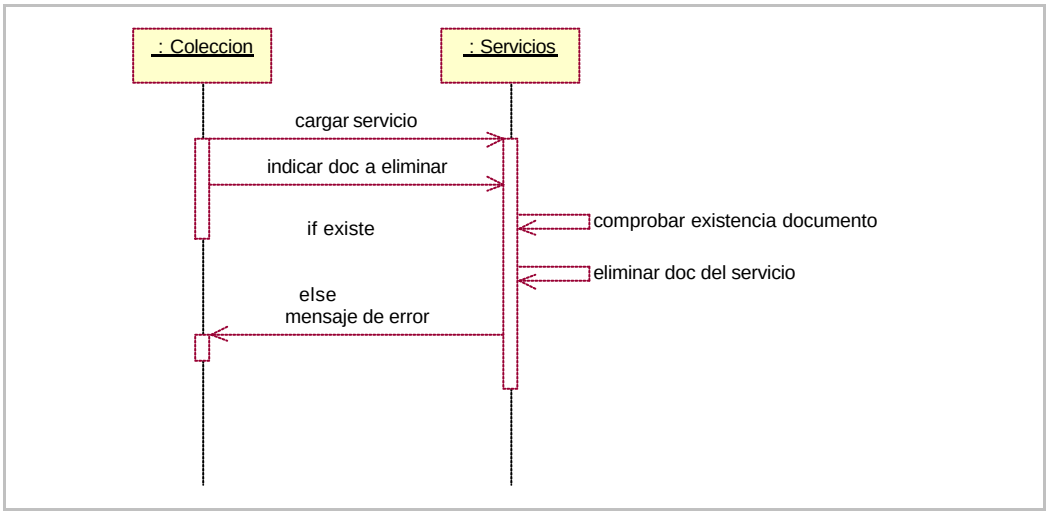


Figura 136: Diseño: Diagrama de secuencia: Eliminar un documento de un servicio sobre grupo de documentos

El diagrama de colaboración de este escenario:

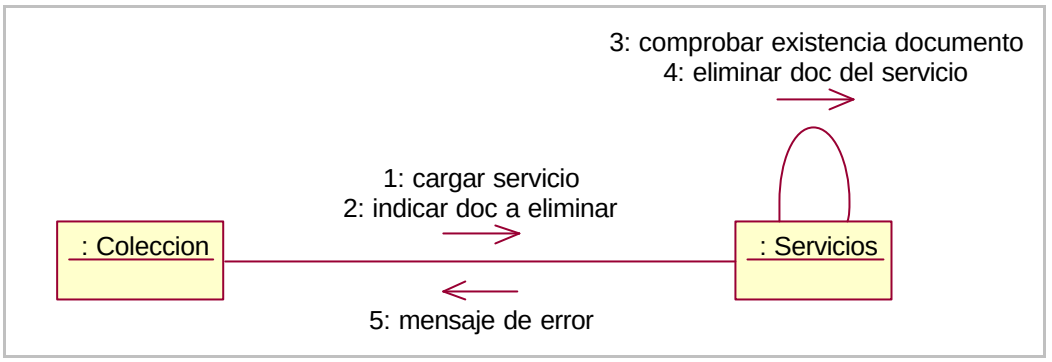


Figura 137: Diseño: Diagrama de colaboración: Eliminar un documento de un servicio sobre grupo de documentos

Escenario: Acceder y realizar búsqueda de documento mediante servicio

Este escenario no cambió con respecto a la fase de análisis, por lo que no se considera necesario volver a repetir los diagramas.

3.2.5.5 Caso de uso: Gestionar presentaciones

Este diagrama de casos de uso se mantiene igual que el de la etapa del análisis, aunque la diferencia con el anterior es que ya no es el usuario el que inicia los escenarios "*Eliminar presentación documento base*" y "*Eliminar presentación documento decorado*", sino que se va a realizar mediante el caso de uso "*Gestionar Colección*" y el caso de uso "*Eliminar Documentos adaptados a la arquitectura*". Ambos se diferencia por: se procede a la eliminación de una presentación de un documento a través "*Eliminar documento adaptado a la arquitectura*" cuando se elimina el documento base y todas sus visiones. De este modo también se eliminan todas las presentaciones de esos documentos. Por otro lado, se eliminarán a través de "*Gestionar Colección*" sin pasar por "*Eliminar documentos adaptados a la arquitectura"* cuando se modifique la colección. Por tanto la precondición que viene a ser común para todos es que el fichero tiene que estar correctamente configurado.

En los diagramas de secuencia y colaboración que aquí se van a describir, es preciso matizar que quien realiza la llamada va a ser el caso de uso "*Gestionar Colección*", "*Eliminar Documentos adaptados a la colección*", "*Gestionar servicios*" y que así se ha hecho en los diagramas, es decir, se parte de la llamada que realiza estos casos de uso para de ese modo conseguir un mayor entendimiento por parte del lector.

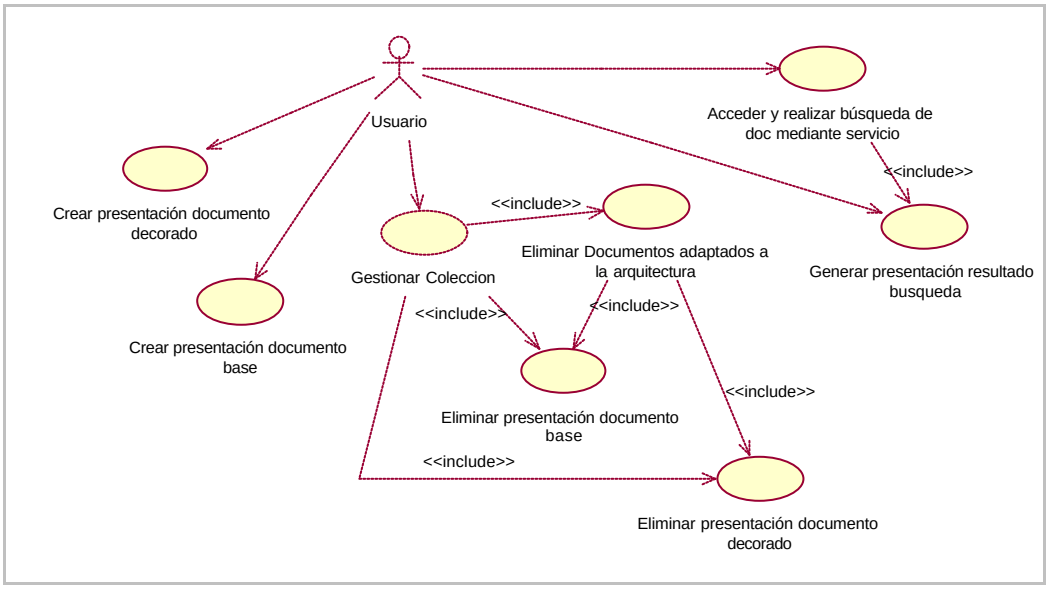


Figura 138: Diseño: Caso de uso: Gestionar presentaciones

Escenario: Generar presentación de un resultado de una búsqueda

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

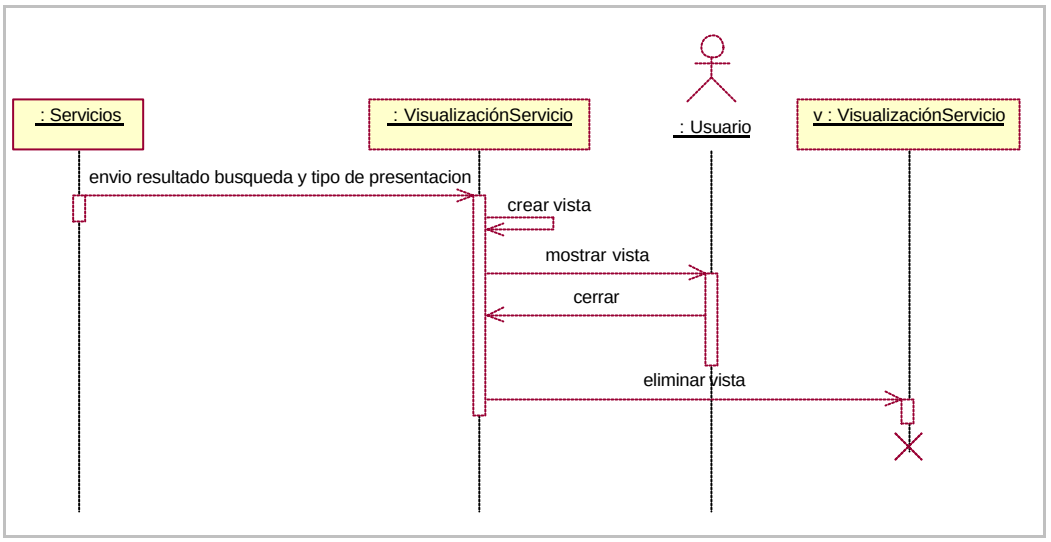


Figura 139: Diseño: Diagrama de secuencia: Generar presentación de un resultado de una búsqueda

El diagrama de colaboración de este escenario:

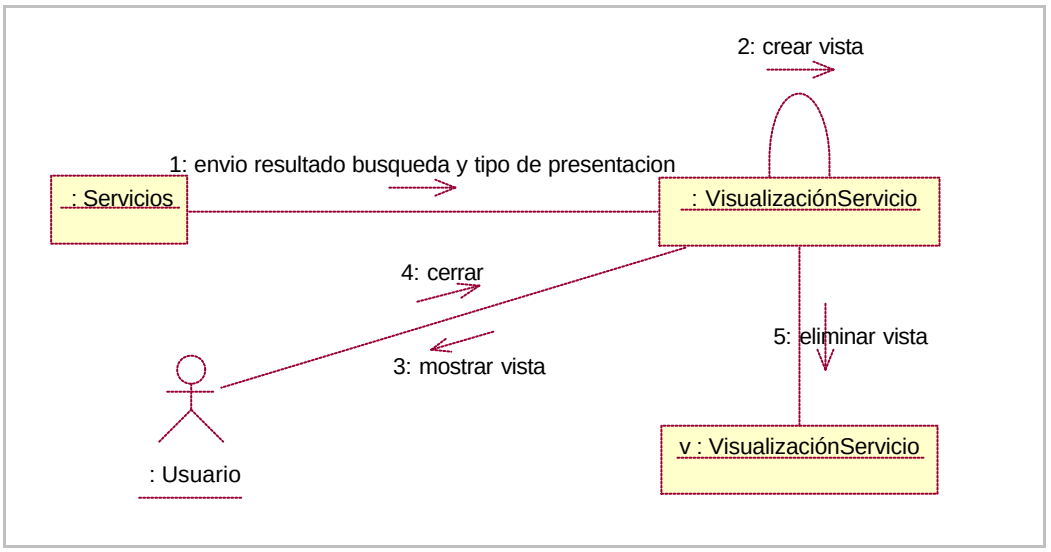


Figura 140: Diseño: Diagrama de colaboración: Generar presentación de un resultado de una búsqueda

Escenario: Crear presentación documento base

Este escenario no ha sufrido ningún cambio, por lo tanto no se considera necesario incluir de nuevo los diagramas de secuencia y colaboración.

Escenario: Crear presentación documento decorado

Este escenario no ha sufrido ningún cambio, por lo tanto no se considera necesario incluir de nuevo los diagramas de secuencia y colaboración.

Escenario: Eliminar presentación documento base

En este escenario se recogen dos situaciones: la primera se produce cuando se procede a "*Eliminar un documento adaptado a la arquitectura*", es decir cuando se elimina un documento base es necesario eliminar su presentación asociada. La segunda sucede cuando se procede a "*Modificar la colección*"; en el caso de que el parámetro que indica el tipo de presentación del documento base haya cambiado, será necesario eliminar el existente.

Vamos a dividir este escenario en dos. Por un lado el procedente de "*Eliminar documento adaptado a la arquitectura*" y por el otro el procedente de "*Gestionar Colección*".

El proceso llevado a cabo por el primer caso se puede ver detalladamente en el siguiente diagrama de secuencia.

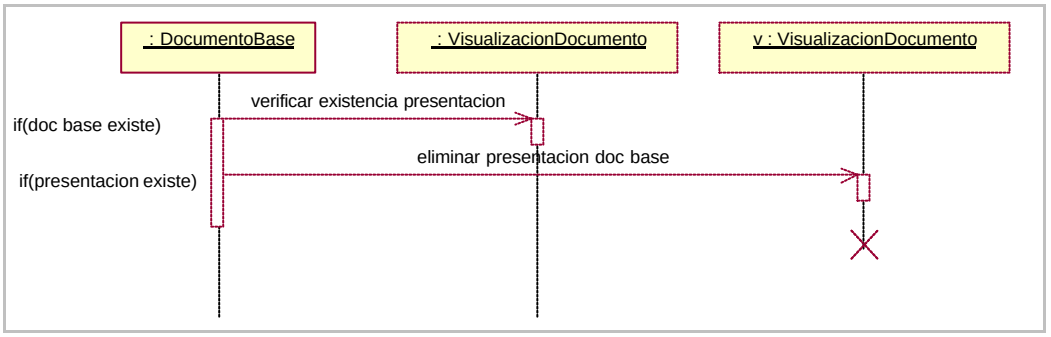


Figura 141: Diseño: Diagrama de secuencia: Eliminar la presentación del documento base1

El diagrama de colaboración de este escenario:

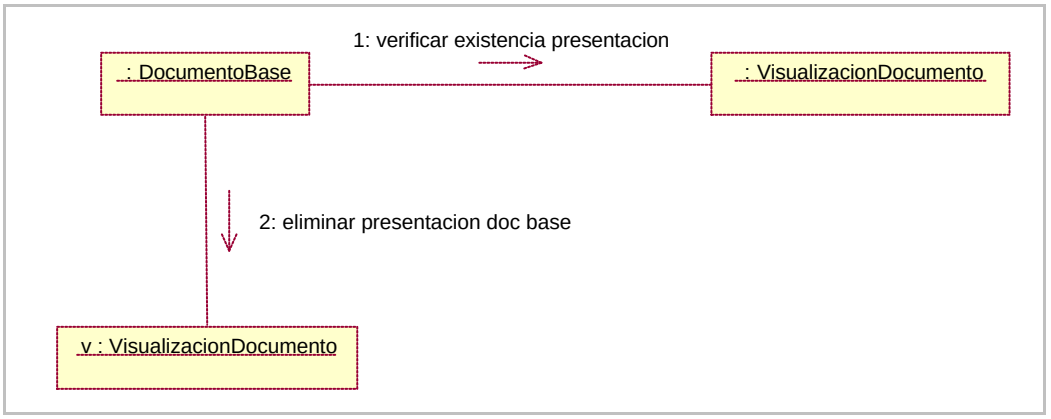


Figura 142: Diseño: Diagrama de colaboración: Eliminar la presentación del documento base1

El proceso llevado a cabo por el segundo caso se puede ver detalladamente en el siguiente diagrama de secuencia.

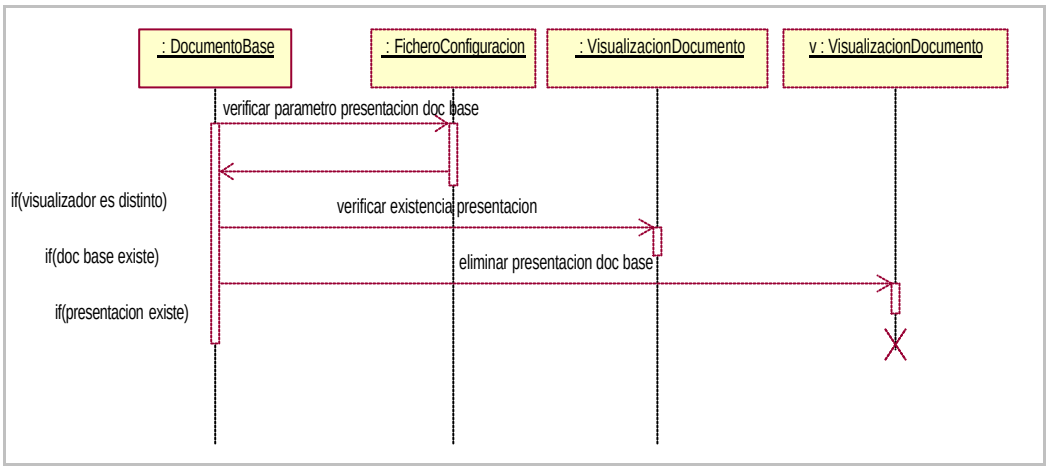


Figura 143: Diseño: Diagrama de secuencia: Eliminar la presentación del documento base2

El diagrama de colaboración de este escenario:

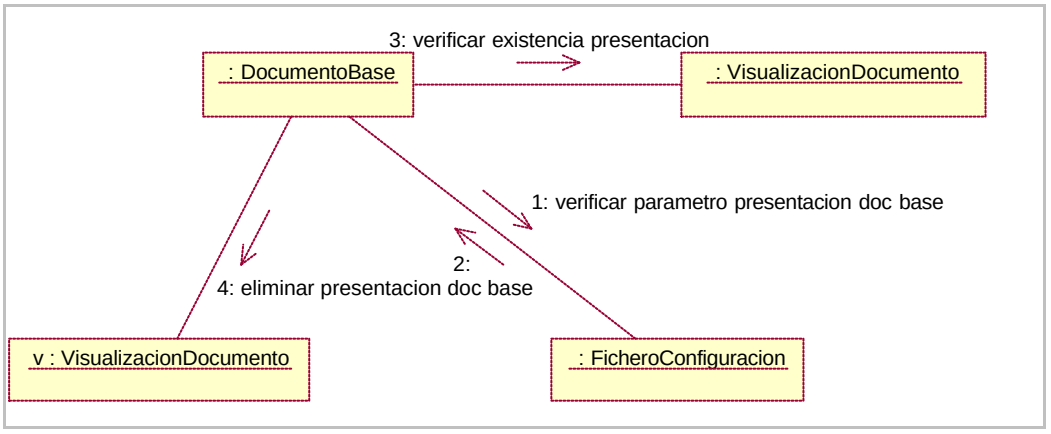


Figura 144: Diseño: Diagrama de colaboración: Eliminar la presentación del documento base2

Escenario: Eliminar la presentación de un documento decorado

En este escenario ocurren dos situaciones: la primera es cuando se procede a "*Eliminar un documento adaptado a la arquitectura*", es decir cuando se elimina un documento decorado es necesario eliminar su presentación. La segunda es cuando se procede a "M*odificar la colección*", en el caso de que el parámetro que indica el tipo de presentación del documento decorado haya cambiado, será necesario eliminar el existente.

Vamos a dividir este escenario en dos. Por un lado el procedente de "*Eliminar documento adaptado a la arquitectura*" y por el otro el procedente de "*Gestionar Colección*".

El proceso llevado a cabo por el primer caso se puede ver detalladamente en el siguiente diagrama de secuencia.

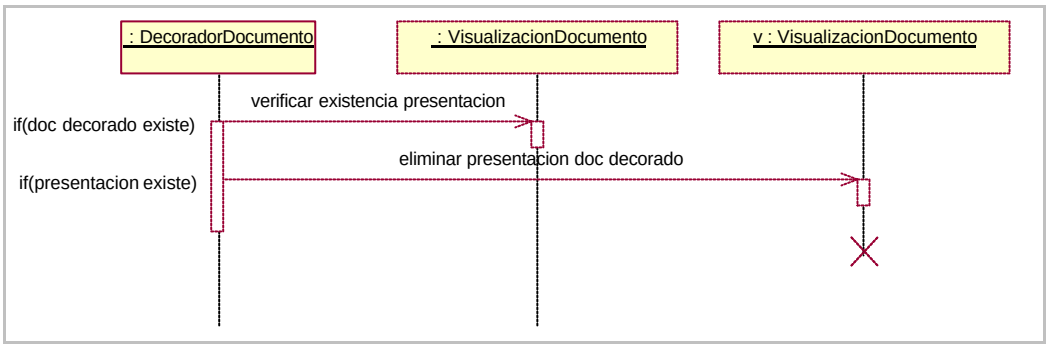


Figura 145: Diseño: Diagrama de secuencia: Eliminar la presentación de un documento decorado1

El diagrama de colaboración de este escenario:

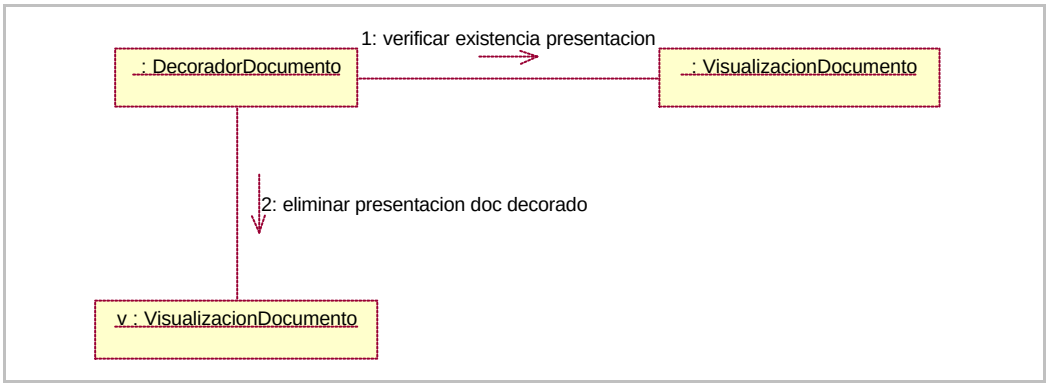


Figura 146: Diseño: Diagrama de colaboración: Eliminar la presentación de un documento decorado1

El proceso llevado a cabo por el segundo caso se puede ver detalladamente en el siguiente diagrama de secuencia.

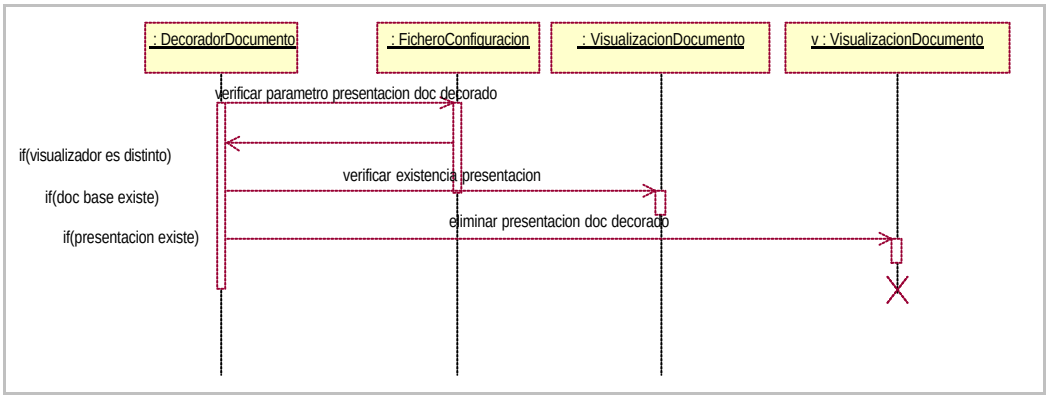


Figura 147: Diseño: Diagrama de secuencia: Eliminar la presentación de un documento decorado2

El diagrama de colaboración de este escenario:

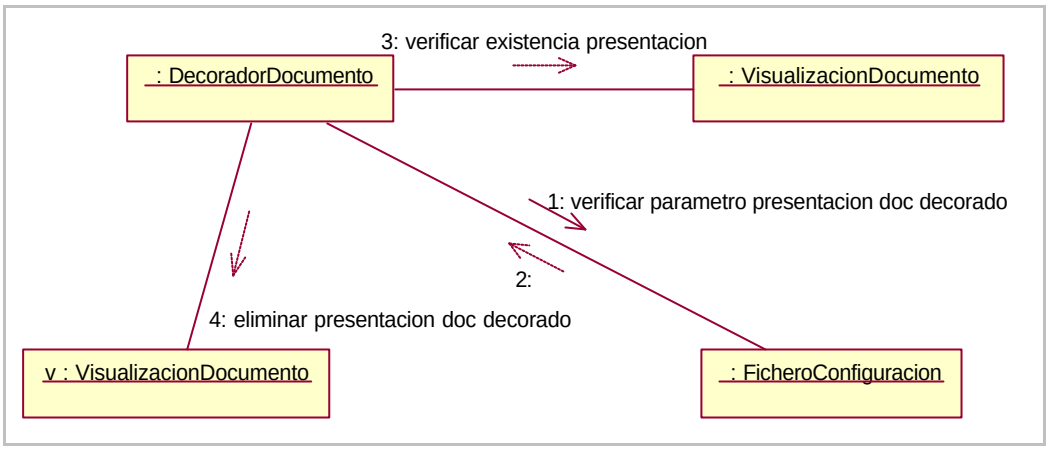


Figura 148: Diseño: Diagrama de colaboración: Eliminar la presentación de un documento decorado2

4. Análisis y Diseño de la Aplicación Arquitex

4.1 Análisis de la aplicación Arquitex

A continuación se procede a detallar las funcionalidades de la aplicación desarrollada para probar la arquitectura, los diagramas de casos de uso asociados y sus diagramas de secuencia. Es necesario indicar que ciertas tareas que se van a realizar ya están detalladas en la arquitectura.

Las funcionalidades que se identificaron en la aplicación desarrollada a mayores de la detallada en la arquitectura se presentan en la siguiente tabla:

FUNCIONALIDADES
26. Modificar el fichero de configuración arquitectura
27. Crear el fichero de configuración de la arquitectura
28. Consultar la ayuda

4.1.1 Diagrama de casos de uso general de la aplicación

El diagrama que aquí se describe es igual que el diagrama que se muestra en la arquitectura, con la diferencia que a este nivel se puede agrupar las funcionalidades del caso de uso "*Crear documentos adaptados a la arquitectura*" y de "*Eliminar los documentos adaptados a la arquitectura*" en uno solo llamado "*Gestionar documentos de la colección*" ya que a nivel de usuario, éste no percibe las tareas que está desempeñando la arquitectura. Otra de las novedades es que se permite configurar el fichero de configuración necesario en la arquitectura desde la propia aplicación que se ha implementado.

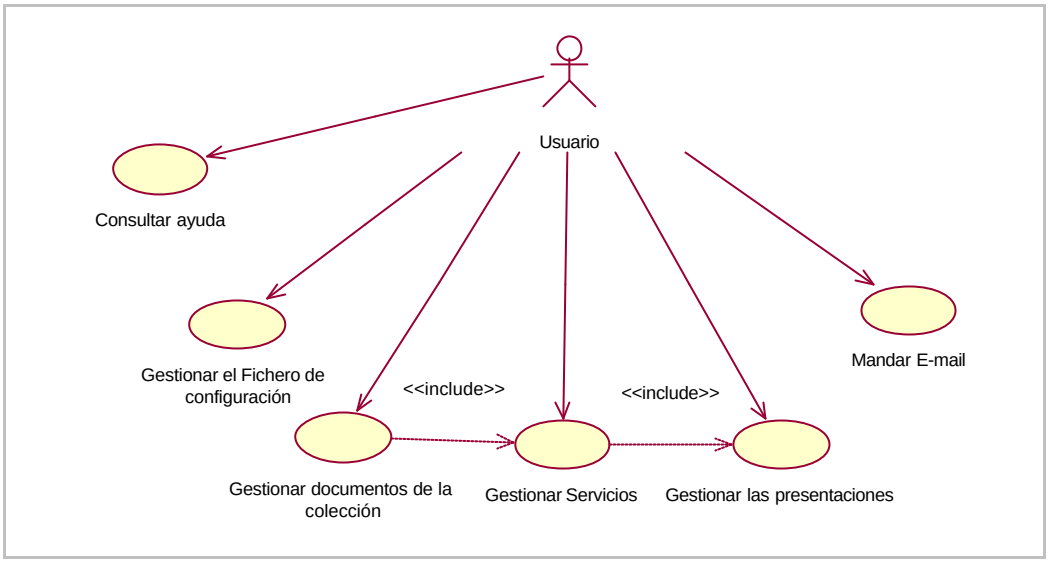


Figura 149: Análisis: Diagrama de casos de uso de la aplicación Arquitex

A continuación se detallan los aspectos de cada uno de estos casos de uso.

4.1.2 Diagrama de casos de uso y de secuencia

4.1.2.1 Caso de uso: Gestionar Documentos de la colección

En este caso de uso es preciso indicar que "*Crear documento adaptados a la arquitectura*", "*Eliminar documento adaptado a la arquitectura*" ya forman parte de la arquitectura que es instanciado por este caso de uso.

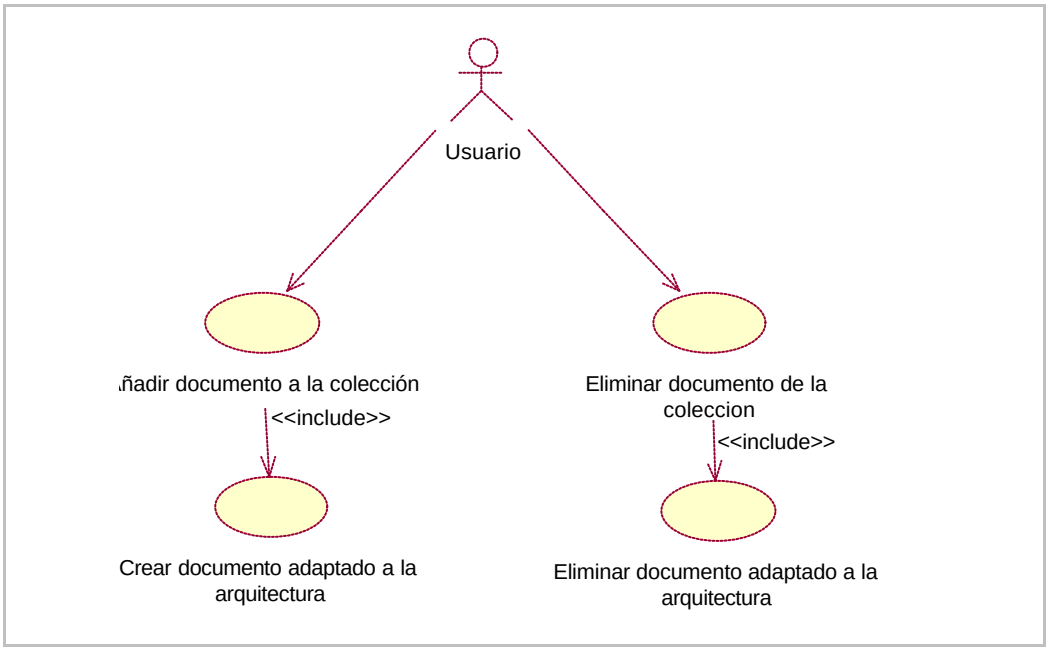


Figura 150: Análisis: Diagrama de casos de uso: Gestionar Documentos de la colección

OBJETIVOS
Este caso de uso permite la funcionalidad poder añadir un nuevo documento al sistema, que puede hacerse introduciendo un documento que será utilizado para generar un fichero XML
ACTORES
Usuario
PRECONDICIONES
El fichero de configuración debe de existir y además debe de estar correctamente configurado. El usuario debe de haber iniciado sesión en la aplicación
FUNCIONES QUE CUMPLE
1, 2, 5, 6, 7, 9, 10, 14, 15, 16, 18, 19, 23, 24, 25

Escenario: Añadir un documento a la colección

DESCRIPCIÓN
Este escenario ocurre cuando el usuario indica al sistema que desea introducir un nuevo documento, creando un documento XML que lo represente.
FLUJO PRINCIPAL
1. El usuario indica al sistema que desea añadir un nuevo documento 2. El sistema habilita un asistente para llevar a cabo dicha tarea 3. El usuario introduce todos los datos 4. El sistema genera el documento XML que representa a dicho documento, el cual posee la estructura adecuada.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: Cancelar adición de un documento]: el usuario cancela prematuramente la operación - [Excepción: Documento ya existente]: el documento que va a ser introducido ya existe en el sistema. - [Excepción: Cancelar la creación del documento XML]: el usuario cancela

prematuramente la operación.

- [Excepción: Documento XML no válido]: el documento XML que ha sido creado, no es válido. Se muestra un mensaje de error.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

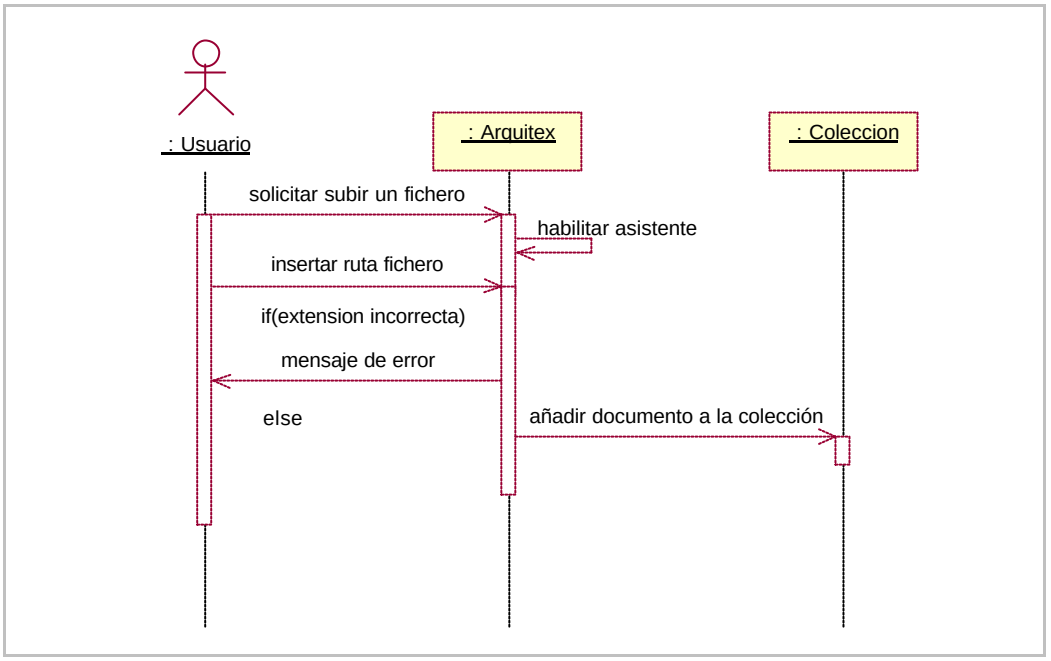


Figura 151: Análisis: Diagrama de secuencia: Añadir un documento a la colección

El diagrama de colaboración de este escenario:

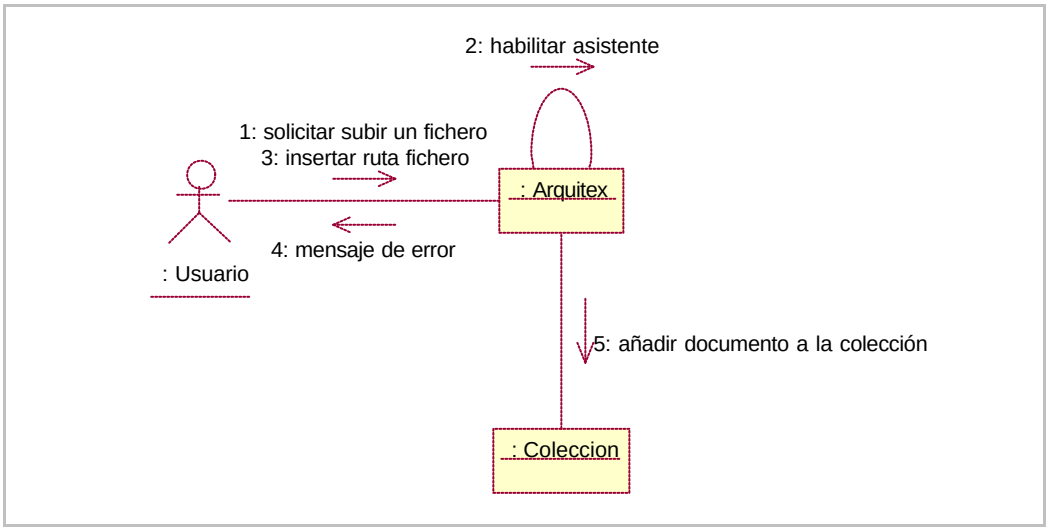


Figura 152: Análisis: Diagrama de colaboración: Añadir un documento a la colección

La operación "*Añadir documento a la colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

Escenario: Eliminar un documento de la colección

DESCRIPCIÓN
Este escenario ocurre cuando el usuario indica al sistema que desea eliminar un documento de la colección, eliminando el documento XML que lo represente, así como todas sus decoraciones como pueden ser resumen, palabras clave, traducciones y clasificación.
FLUJO PRINCIPAL
1. El usuario indica al sistema que desea eliminar un documento 2. El sistema habilita un asistente para llevar a cabo dicha tarea 3. El usuario confirma que quiere eliminarlo del sistema 4. El sistema eliminar el documento XML que representa a dicho documento, así como las decoraciones que posee.
PRECONDICIONES
Para poder eliminar un documento, es necesario que existan documentos en la colección previamente.
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: Cancelar eliminación de un documento]: el usuario cancela prematuramente la operación - [Excepción: Documento ya borrado]: el documento que va a ser eliminado de la colección no existe.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

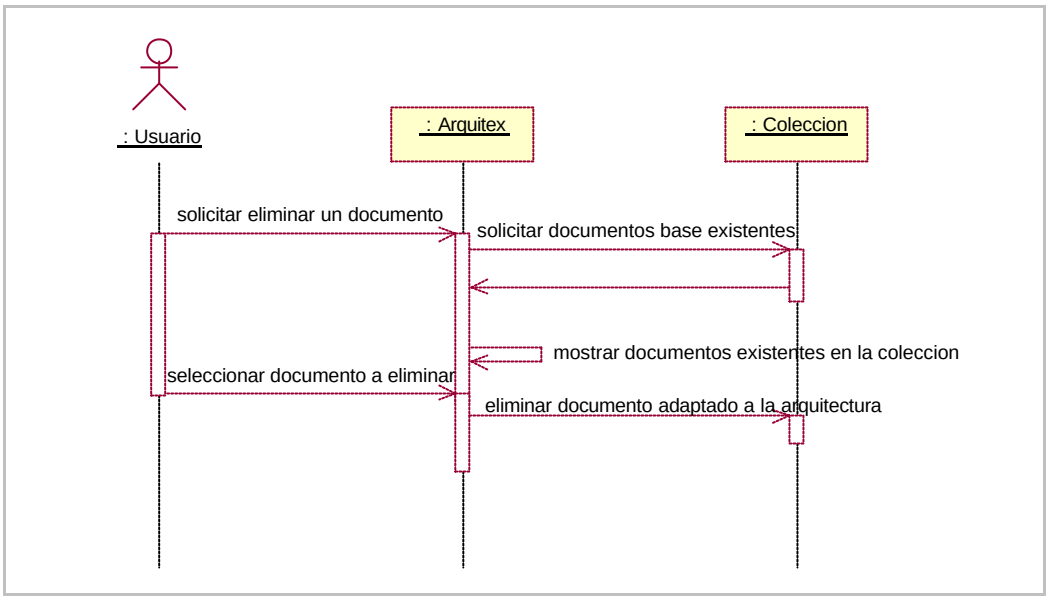


Figura 153: Análisis: Diagrama de secuencia: Eliminar un documento de la colección

El diagrama de colaboración de este escenario:

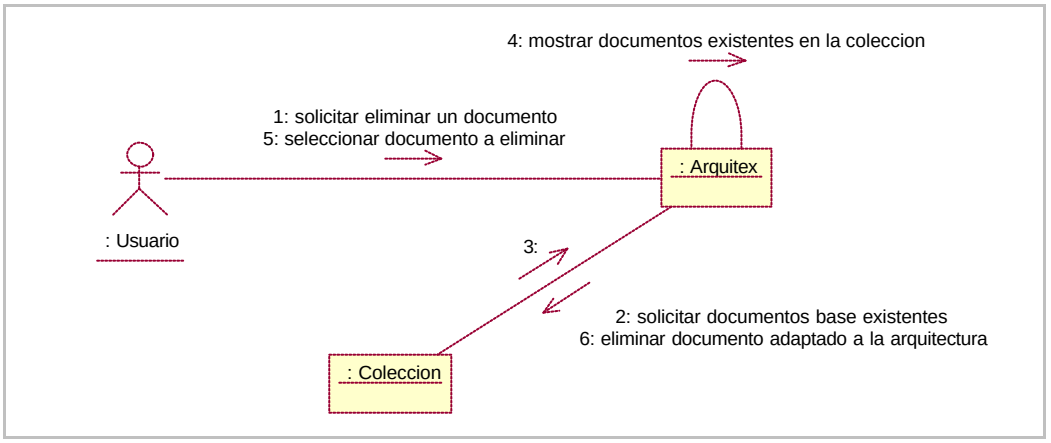


Figura 154: Análisis: Diagrama de colaboración: Eliminar un documento de la colección

La operación "*Eliminar documento a la colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

4.1.2.2 Caso de uso: Consultar ayuda

Este caso de uso es además un escenario propiamente dicho por lo que no se realiza el diagrama de casos de uso sino solo el escenario.

OBJETIVOS
Este caso de uso permite la funcionalidad de poder consultar la ayuda que proporciona la aplicación Arquitex. En ella se podrá encontrar la información proporcionada en el manual de usuario de este proyecto.

ACTORES
Usuario
PRECONDICIONES
El usuario debe de haber iniciado sesión en Arquitex
FUNCIONES QUE CUMPLE
28

Escenario: Consultar ayuda

DESCRIPCIÓN
Este escenario ocurre cuando el usuario indica al sistema que desea consultar la ayuda de la herramienta Arquitex.
FLUJO PRINCIPAL
1. El usuario indica al sistema que desea consultar la ayuda 2. El sistema muestra el documento
FLUJO EXCEPCIONAL DE EVENTOS
Se distinguen los siguientes: - [Excepción: No existe el documento de ayuda]: el documento de ayuda no se encuentra

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

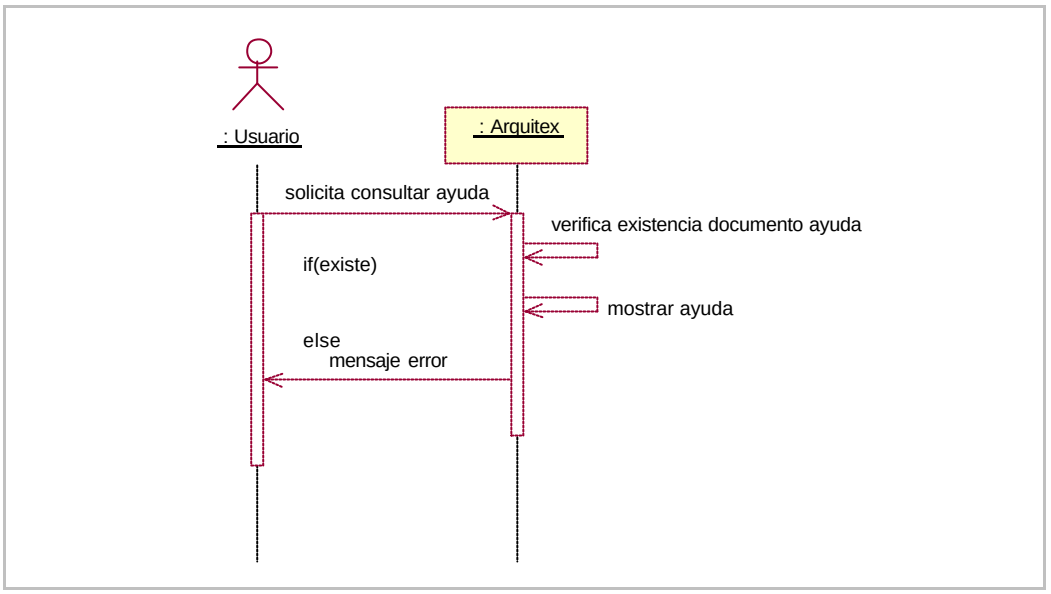


Figura 155: Análisis: Diagrama de secuencia: Consultar ayuda

El diagrama de colaboración de este escenario:

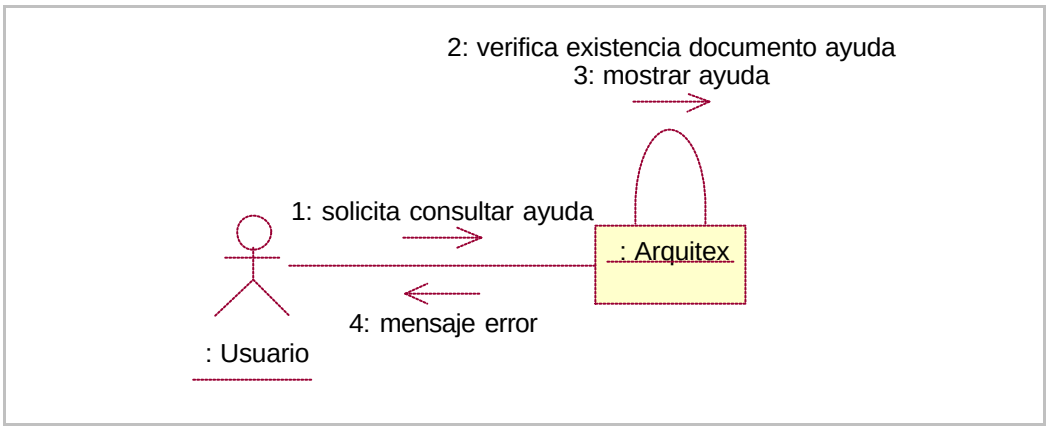


Figura 156: Análisis: Diagrama de colaboración: Consultar ayuda

4.1.2.3 Caso de uso: Gestionar el fichero de configuración

En este caso de uso es preciso indicar que "*Inicializar Colección*"" y "*Modificar Colección*" ya forman parte de la arquitectura que es instanciado por este caso de uso.

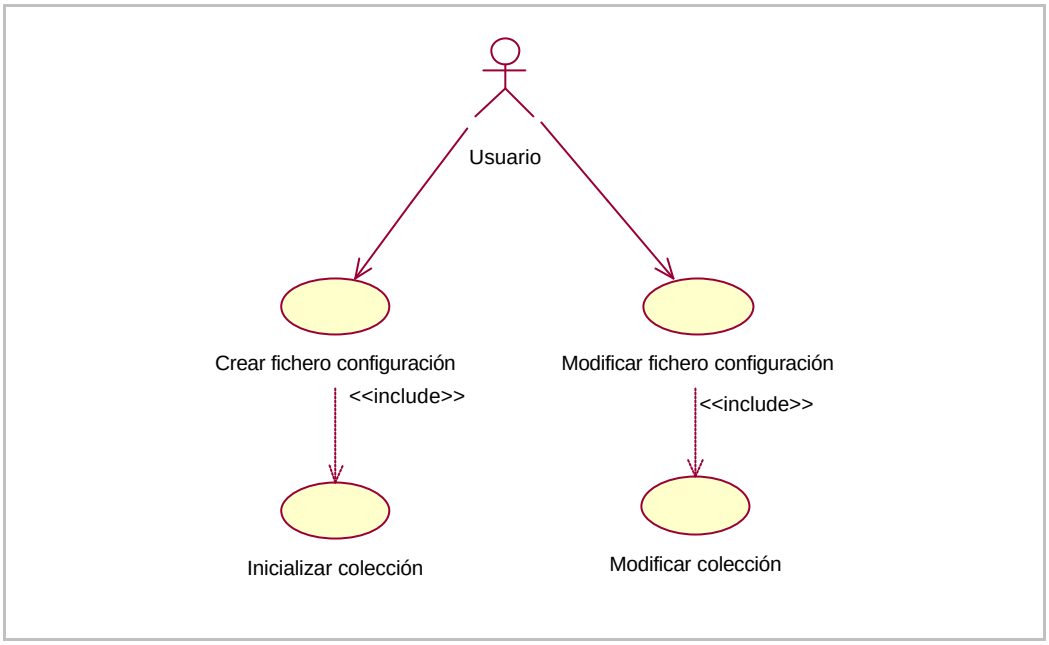


Figura 157: Análisis: Diagrama de casos de uso: Gestionar el fichero de configuración

OBJETIVOS
Este caso de uso permite que el usuario pueda crear el fichero de configuración en el caso de que este no exista; en caso contrario, si el fichero de configuración existe, se podrá modificar su contenido.
ACTORES
Usuario
PRECONDICIONES
El usuario debe de haber iniciado sesión en la aplicación
POSTCONDICIONES
El fichero está creado y además está correctamente configurado La colección está cargada.
FUNCIONES QUE CUMPLE
1,2,5,6,7,8,9,10,14,15,16,17,18,19,20,21, 22, 26, 27

Escenario: Crear fichero de configuración

DESCRIPCIÓN
Este escenario consiste en crear el fichero de configuración cuando éste no existe o bien cuando al hacer la comprobación de si el fichero existe, devuelve algún

tipo de error debido a qué por ejemplo le falta alguno de los campos.
FLUJO PRINCIPAL
1. El sistema indica que el fichero de configuración no existe o bien que éste es erróneo. 2. El usuario indica si desea realmente crear el fichero 3. El usuario indica los directorios donde se van a almacenar los documentos, las decoraciones y los servicios. 4. El usuario indica cuales van a ser las clases cargadoras de documentos 5. El usuario indica los nombres de los resumidores (si existen), de los traductores (si existen), de los clasificadores (si existen) y de los extractores (si existen). 6. El usuario indica los nombres de los indexadores (si existen) y de los clusterings (si existen). 7. El usuario indica los nombres de las cadenas de servicios (si existen) 8. El usuario acepta esos cambios 9. Se produce algún tipo de excepción: 9.1 Indique la dirección donde almacenar los documentos base 9.2 Indique la dirección donde almacenar las decoraciones de los documentos base 9.3 Indique la dirección donde almacenar los servicios 9.4 Indique al menos un cargador 9.5 Indique al menos un servicio de documentos individuales 9.6 Indique los servicios de grupos de documentos 10. Por cada uno de los resumidores introducidos, el usuario tiene que indicar la clase que la implementa, la fuente de la que va a hacer uso, y el parámetro asociado que es el numero de líneas máximo del resumen 11. Por cada uno de los extractores introducidos, el usuario tiene que indicar la clase que la implementa y la fuente de la que va a hacer uso 12. Por cada uno de los clasificadores introducidos, el usuario tiene que indicar la clase que la implementa y la fuente de la que va a hacer uso 13. Por cada uno de los traductores introducidos, el usuario tiene que indicar la clase que la implementa, la fuente de la que va a hacer uso, el idioma origen de la fuente y el idioma destino del documento. Éstos son parámetros de la clase implementada. 14. Por cada uno de los indexadores y clusterings introducidos, el usuario tiene que indicar la clase que la implementa, la fuente de la que va a hacer uso, indicar si va a ser temporal o no, y las clases que implementan la vista de

documentos y la vista de listas de documentos

15. Por cada una de las cadenas de servicios introducidos, el usuario tiene que indicar el numero de pasos del que va a constar
16. El usuario acepta esos cambios
17. Se produce algún tipo de excepción:
    - 17.1 Error: Es necesario cubrir todos los campos del formulario
    - 17.2 Error: … no es una clase
    - 17.3 Error: el campo temporal de los servicios de grupos de documentos solo puede ser verdadero o falso
    - 17.4 Indique al menos un cargador
    - 17.5 Indique al menos un servicio de documentos individuales
    - 17.6 Indique los servicios de grupos de documentos
18. Por cada uno de las cadenas de servicios introducidos, el usuario tiene que indicar el los pasos del que va a constar.
19. En caso de haber insertado en los servicios de documentos individuales alguna fuente que no sea ni DocumentoBase ni algún servicio existente entre los resumidores, extractores, clasificadores o traductores, será necesario cumplimentar cual es la clase que la implementa, cual es la fuente de la que va a hacer uso, indicar de que tipo es y dependiendo de si es traductor, indicará el origen y el destino; si es resumidor, indicará el numero de líneas como máximo

FLUJO EXCEPCIONAL DE EVENTOS

- [Indique la dirección donde almacenar los documentos base]: ocurre cuando no se indica ninguna ruta en el campo de directorio de documentos base
- [Indique la dirección donde almacenar las decoraciones de los documentos base]: ocurre cuando no se indica ninguna ruta en el campo de directorio de decoraciones
- [Indique la dirección donde almacenar los servicios]: ocurre cuando no se indica ninguna ruta en el campo de directorio de servicios
- [Indique al menos un cargador]: ocurre cuando no se indica ningún cargador de documentos
- [Indique al menos un servicio de documentos individuales]: ocurre cuando no se indica ningún servicio de documentos individuales
- [Indique los servicios de grupos de documentos] : ocurre cuando no se indica ningún servicio de grupos de documentos
- [Error: Es necesario cubrir todos los campos del formulario]: falta algún campo por rellenar

- [Error: … no es una clase]: el sistema estaba esperando que se introdujese el nombre de una clase existente
- [Error: el campo temporal de los servicios de grupos de documentos solo puede ser verdadero o falso]

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

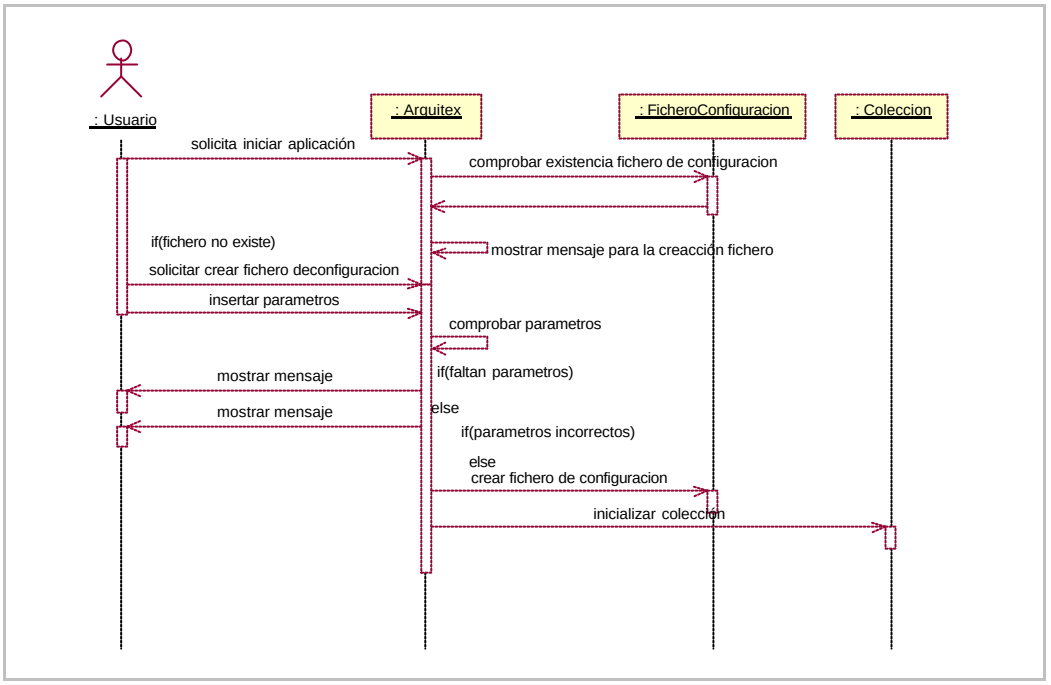


Figura 158: Análisis: Diagrama de secuencia: Crear fichero de configuración

El diagrama de colaboración de este escenario:

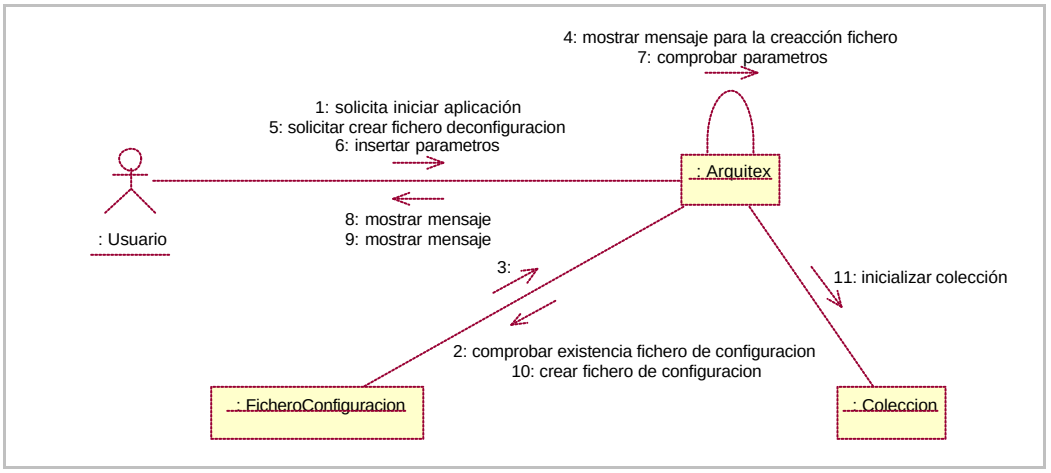


Figura 159: Análisis: Diagrama de colaboración: Crear fichero de configuración

La operación "*Inicializar la colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

Escenario: Modificar el fichero de configuración

DESCRIPCIÓN
Este escenario ocurre cuando el usuario desea modificar el fichero para cambiar los parámetros que este posee, o cuando éste es incorrecto.
FLUJO PRINCIPAL
1. El usuario indica que desea modificar el fichero 2. El sistema muestra los directorios donde se almacenan los documentos base, las decoraciones y los servicios. Muestra las clases cargadoras de documentos. Muestra los nombres de los servicios sobre documentos individuales como son los resumidores, extractores, clasificadores y traductores si existen. Muestra los nombres de los servicios sobre grupos de documentos como son los indexadores y clusterings si existen. Finalmente muestra los nombres de las cadenas de servicios si existen. 3. El usuario modifica los directorios donde se van a almacenar los documentos, las decoraciones y los servicios. 4. El usuario modifica cuales van a ser las clases cargadoras de documentos 5. El usuario modifica los nombres de los resumidores (si existen), de los traductores (si existen), de los clasificadores (si existen) y de los extractores (si existen). 6. El usuario modifica los nombres de los indexadores (si existen) y de los clusterings (si existen). 7. El usuario modifica los nombres de las cadenas de servicios (si existen) 8. El usuario acepta esos cambios 9. Se produce algún tipo de excepción: 9.1 Indique la dirección donde almacenar los documentos base 9.2 Indique la dirección donde almacenar las decoraciones de los documentos base 9.3 Indique la dirección donde almacenar los servicios 9.4 Indique al menos un cargador 9.5 Indique al menos un servicio de documentos individuales 9.6 Indique los servicios de grupos de documentos 10. Por cada uno de los resumidores introducidos, el usuario tiene que indicar la clase que la implementa, la fuente de la que va a hacer uso, y el parámetro asociado que es el numero de líneas máximo del resumen en caso de no existir ya. Sino solo tiene que modificar la existente 11. Por cada uno de los extractores introducidos, el usuario tiene que indicar la clase que la implementa y la fuente de la que va a hacer uso en caso de no

existir ya. Sino solo tiene que modificar la existente

12. Por cada uno de los clasificadores introducidos, el usuario tiene que indicar la clase que la implementa y la fuente de la que va a hacer uso en caso de no existir ya. Sino solo tiene que modificar la existente
13. Por cada uno de los traductores introducidos, el usuario tiene que indicar la clase que la implementa, la fuente de la que va a hacer uso, el idioma origen de la fuente y el idioma destino del documento (éstos son parámetros de la clase implementada) en caso de no existir ya. Sino solo tiene que modificar la existente.
14. Por cada uno de los indexadores y clusterings introducidos, el usuario tiene que indicar la clase que la implementa, la fuente de la que va a hacer uso, indicar si va a ser temporal o no, y las clases que implementan la vista de documentos y la vista de listas de documentos en caso de no existir ya. Sino solo tiene que modificar la existente
15. Por cada una de las cadenas de servicios introducidos, el usuario tiene que indicar el número de pasos del que va a constar en caso de no existir ya. Sino solo tiene que modificar la existente
16. El usuario acepta esos cambios
17. Se produce algún tipo de excepción:
    - 17.1 Error: Es necesario cubrir todos los campos del formulario
    - 17.2 Error: … no es una clase
    - 17.3 Error: el campo temporal de los servicios de grupos de documentos solo puede ser verdadero o falso
    - 17.4 Indique al menos un cargador
    - 17.5 Indique al menos un servicio de documentos individuales
    - 17.6 Indique los servicios de grupos de documentos
18. Por cada uno de las cadenas de servicios introducidos, el usuario tiene que indicar el los pasos del que va a constar en caso de no existir ya. Sino solo tiene que modificar la existente
19. En caso de haber insertado en los servicios de documentos individuales alguna fuente que no sea ni DocumentoBase ni algún servicio existente entre los resumidores, extractores, clasificadores o traductores, será necesario cumplimentar cual es la clase que la implementa, cual es la fuente de la que va a hacer uso, indicar de que tipo es y dependiendo de si es traductor, indicará el origen y el destino; si es resumidor, indicará el numero de líneas como máximo
20. Se produce algún tipo de excepción:

FLUJO EXCEPCIONAL DE EVENTOS

- [Indique la dirección donde almacenar los documentos base]: ocurre

cuando no se indica ninguna ruta en el campo de directorio de documentos base

- [Indique la dirección donde almacenar las decoraciones de los documentos base]: ocurre cuando no se indica ninguna ruta en el campo de directorio de decoraciones
- [Indique la dirección donde almacenar los servicios]: ocurre cuando no se indica ninguna ruta en el campo de directorio de servicios
- [Indique al menos un cargador]: ocurre cuando no se indica ningún cargador de documentos
- [Indique al menos un servicio de documentos individuales]: ocurre cuando no se indica ningún servicio de documentos individuales
- [Indique los servicios de grupos de documentos] : ocurre cuando no se indica ningún servicio de grupos de documentos
- [Error: Es necesario cubrir todos los campos del formulario]: falta algún campo por rellenar
- [Error: … no es una clase]: el sistema estaba esperando que se introdujese el nombre de una clase existente
- [Error: el campo temporal de los servicios de grupos de documentos solo puede ser verdadero o falso]

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

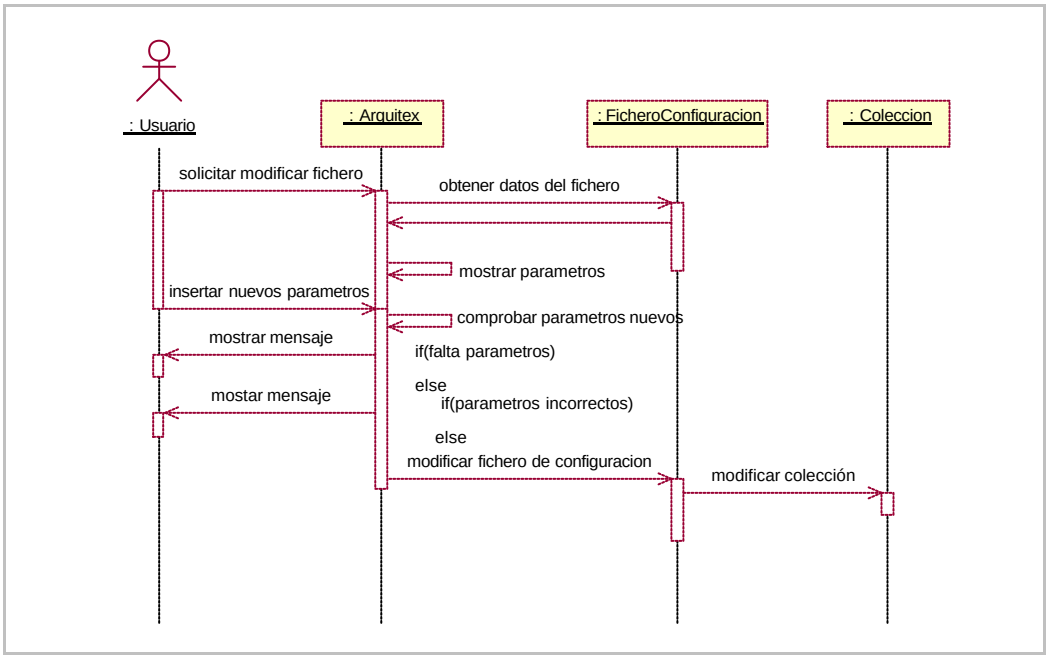


Figura 160: Análisis: Diagrama de secuencia: Modificar fichero de configuración

El diagrama de colaboración de este escenario:

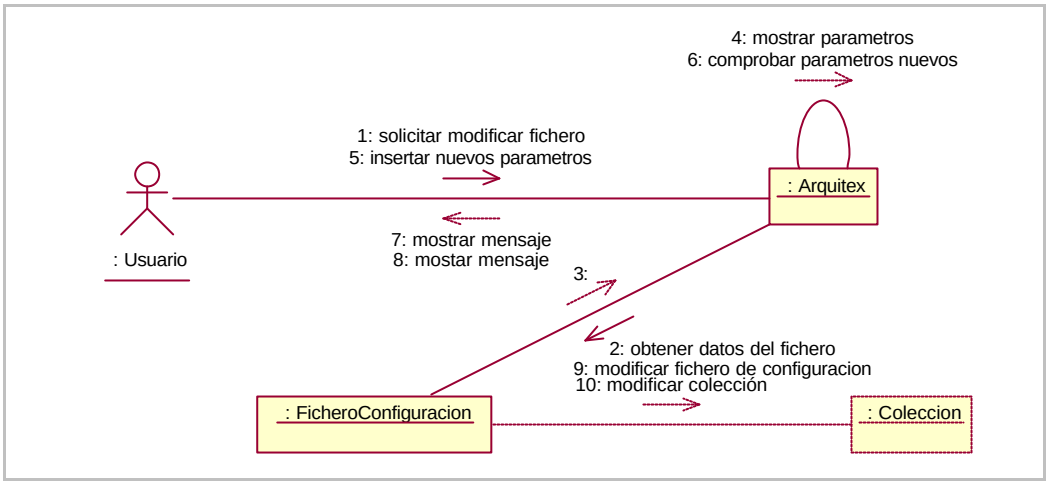


Figura 161: Análisis: Diagrama de colaboración: Modificar fichero de configuración

La operación "*Modificar colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

4.1.2.4 Caso de uso: Gestionar las presentaciones

En este caso de uso es preciso indicar que "*Crear presentación documento base*" y "*Crear presentación documento decorado*" ya forman parte de la arquitectura que es instanciado por este caso de uso.

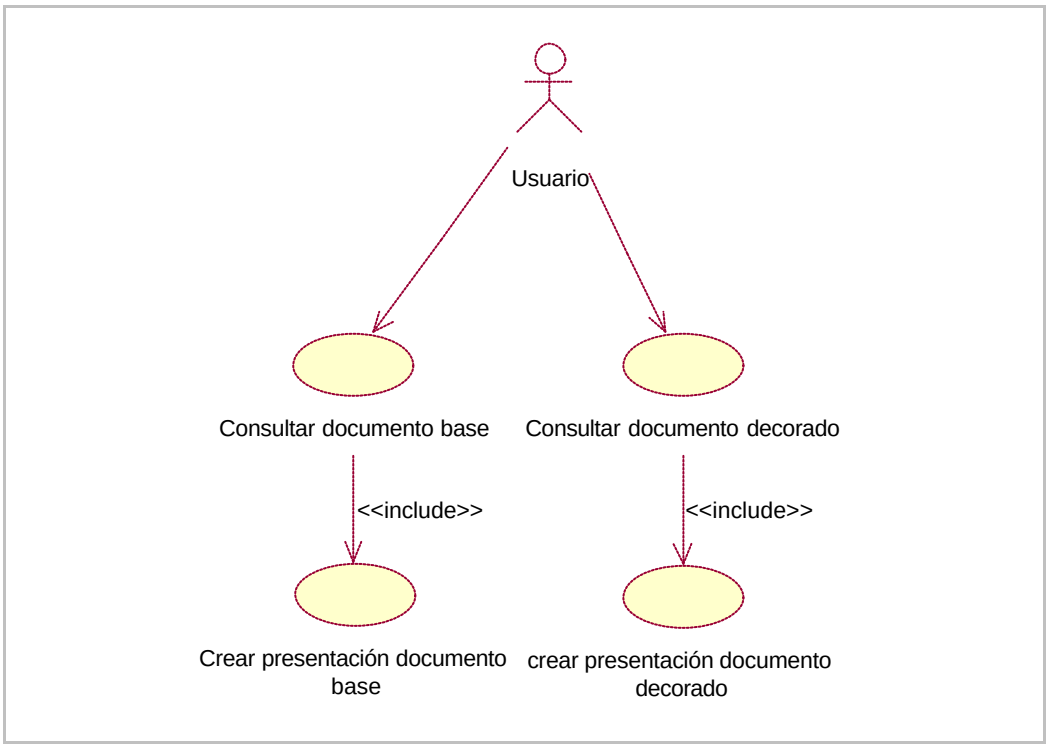


Figura 162: Análisis: Diagrama de casos de uso: Gestionar las presentaciones

OBJETIVOS
Este caso de uso permite al usuario consultar tanto el resultado obtenido tras realizar una búsqueda, como consultar los documentos base o consultar un documento decorado.
ACTORES
Usuario
PRECONDICIONES
El usuario debe de haber iniciado sesión en la aplicación. El fichero tiene que estar correctamente configurado.
FUNCIONES QUE CUMPLE
11,12,13

Escenario: Consultar documento base

DESCRIPCIÓN
Este escenario ocurre después de haber realizado una búsqueda, el usuario decide visualizar uno de los documentos que forman parte del resultado de la búsqueda.
PRECONDICIONES
La fuente del servicio que realizó la búsqueda, tiene que ser un documento base. El resultado de la búsqueda tiene que haber devuelto al menos un documento.
FLUJO PRINCIPAL
1. El sistema muestra el resultado de la búsqueda que le indicó el usuario 2. El usuario selecciona un documento de la lista 3. El sistema verifica que existe la presentación asociada a ese documento base 4. El sistema muestra el contenido del documento
FLUJO EXCEPCIONAL DE EVENTOS
- [No existen documentos asociados a esta búsqueda]: el resultado de la búsqueda devolvió 0 documentos.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

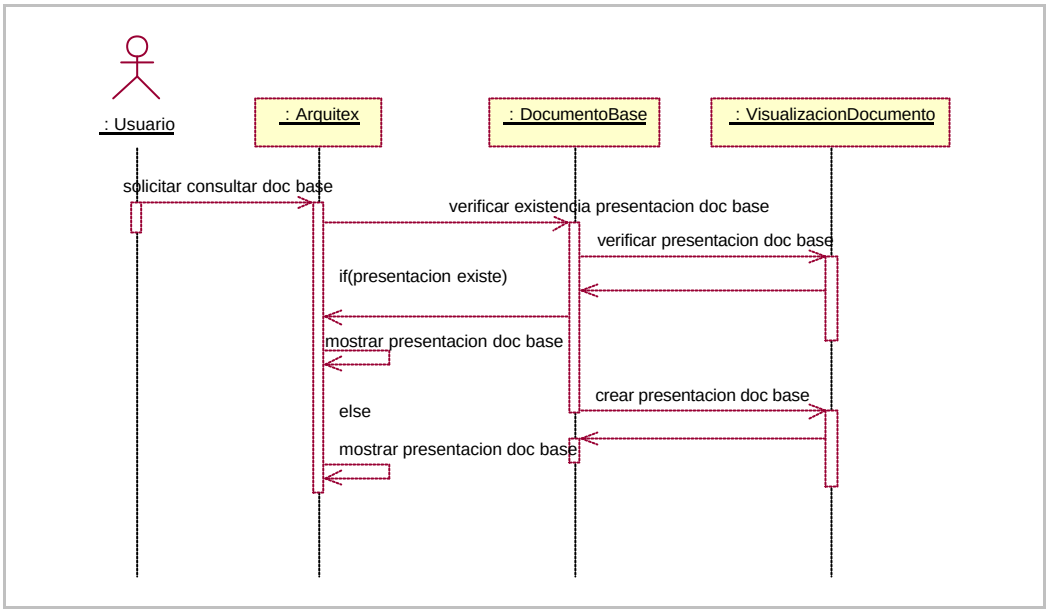


Figura 163: Análisis: Diagrama de secuencia: Consultar documento base

El diagrama de colaboración de este escenario:

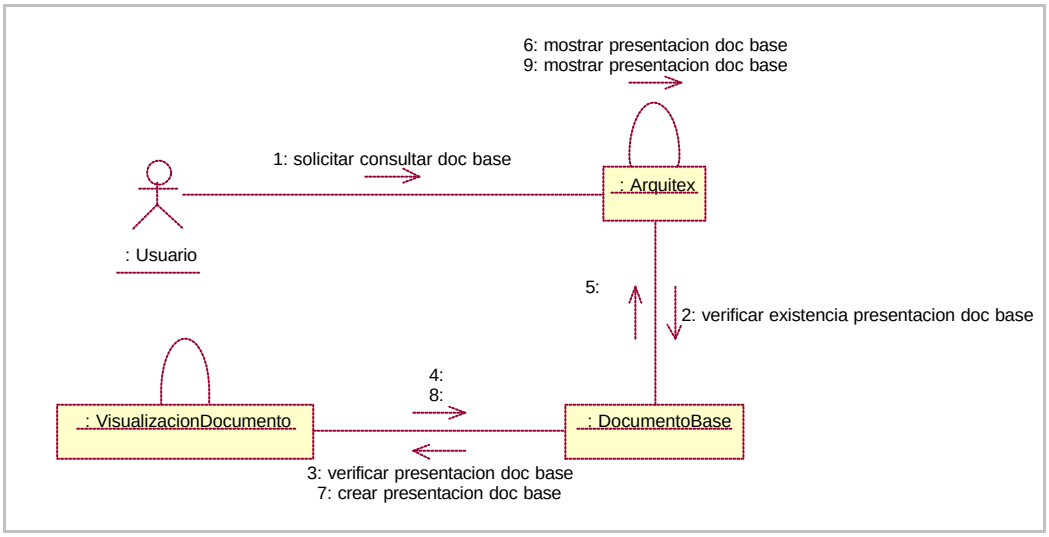


Figura 164: Análisis: Diagrama de colaboración: Consultar documento base

La operación "*Crear presentación documento base*" forma parte ya de la arquitectura. Para más información consulte el análisis y diseño de la arquitectura en la página 129.

Escenario: Consultar documento decorado

DESCRIPCIÓN
Este escenario ocurre después de haber realizado una búsqueda, el usuario decide visualizar uno de los documentos que forman parte del resultado de la

búsqueda.
PRECONDICIONES
La fuente del servicio que realizó la búsqueda, tiene que ser un documento decorado. El resultado de la búsqueda tiene que haber devuelto al menos un documento.
FLUJO PRINCIPAL
1. El sistema muestra el resultado de la búsqueda que le indicó el usuario 2. El usuario selecciona un documento de la lista 3. El sistema verifica que existe la presentación asociada a ese documento decorado 4. El sistema muestra el contenido del documento
FLUJO EXCEPCIONAL DE EVENTOS
- [No existen documentos asociados a esta búsqueda]: el resultado de la búsqueda devolvió 0 documentos.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

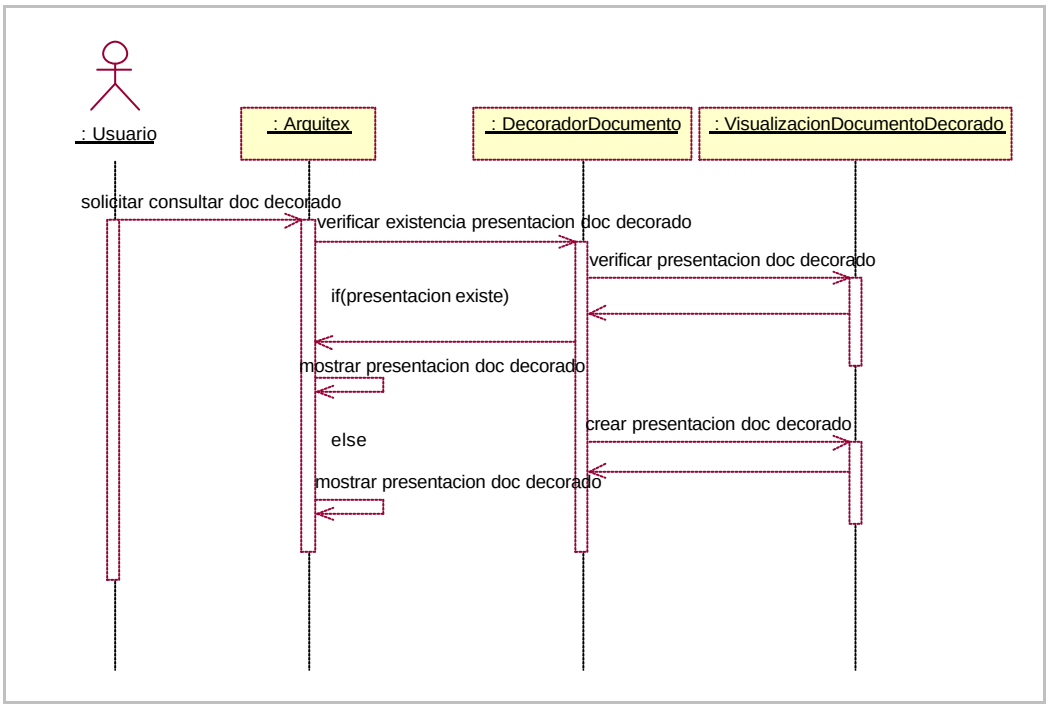


Figura 165: Análisis: Diagrama de secuencia: Consultar documento decorado

El diagrama de colaboración de este escenario:

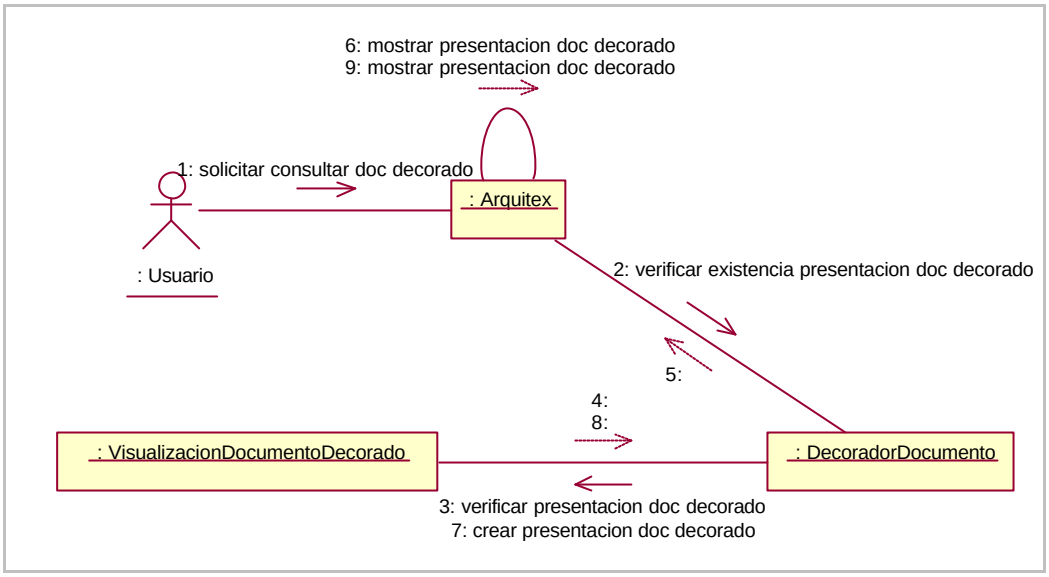


Figura 166: Análisis: Diagrama de colaboración: Consultar documento decorado

La operación *"Crear presentación documento decorado"* forma parte ya de la arquitectura. Para más información consulte el análisis y diseño de la arquitectura en la página 129.

4.1.2.5 Caso de uso: Gestionar Búsquedas

En este caso de uso es preciso indicar que *"Acceder y realizar búsqueda de documentos mediante servicio"* ya forman parte de la arquitectura que es instanciado por este caso de uso.

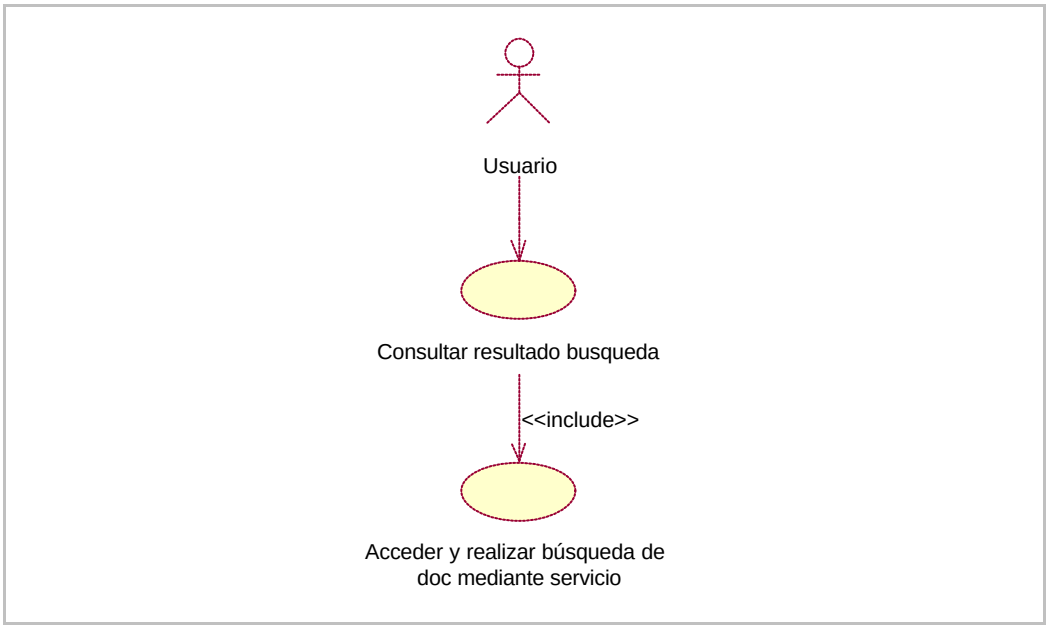


Figura 167: Análisis: Diagrama de casos de uso: Gestionar Búsquedas

OBJETIVOS
Este caso de uso se inicia cuando el usuario solicita realizar una búsqueda y visualizar el resultado del mismo.
ACTORES
Usuario
PRECONDICIONES
El usuario debe de haber iniciado sesión en la aplicación. El fichero tiene que estar correctamente configurado.
FUNCIONES QUE CUMPLE
3, 4

Escenario: Consultar resultado de la búsqueda

DESCRIPCIÓN
Este escenario ocurre cuando el usuario indica la consulta al servicio que va a realizar la consulta.
FLUJO PRINCIPAL
1. El selecciona realizar una búsqueda a través de un servicio 2. El usuario inserta la consulta al servicio 3. El sistema realiza la búsqueda a través del servicio seleccionado 4. El sistema muestra la lista de documentos que cumplen con la consulta
FLUJO EXCEPCIONAL DE EVENTOS
- [No existen documentos asociados a esta búsqueda]: el resultado de la búsqueda devolvió 0 documentos.

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

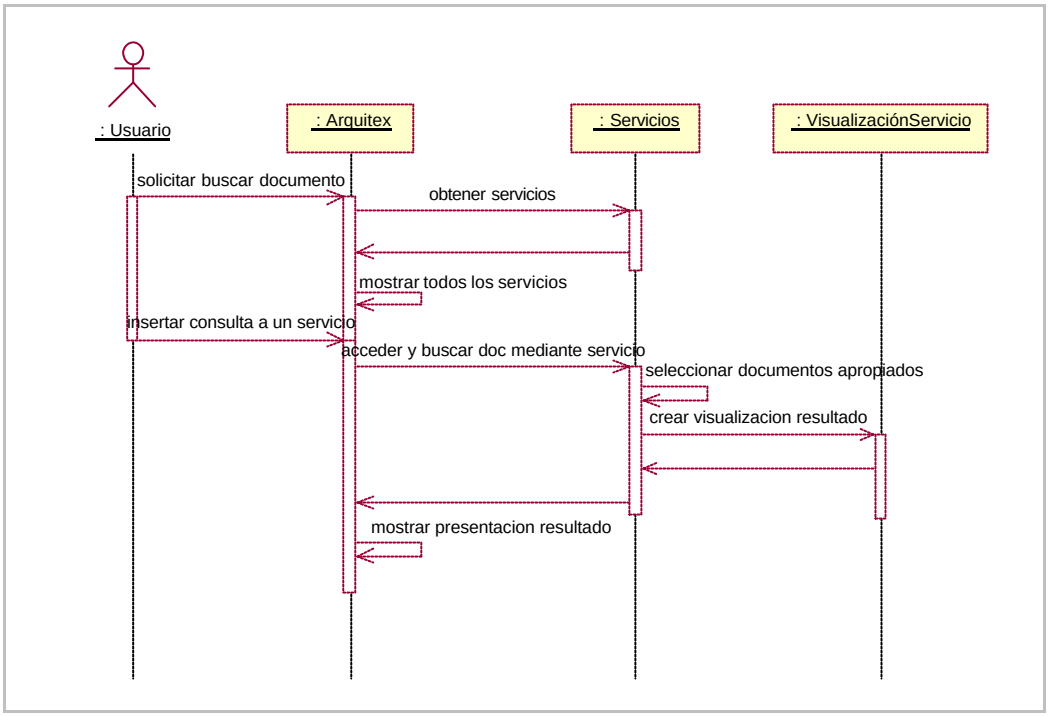


Figura 168: Análisis: Diagrama de secuencia: Consultar resultado de la búsqueda

El diagrama de colaboración de este escenario:

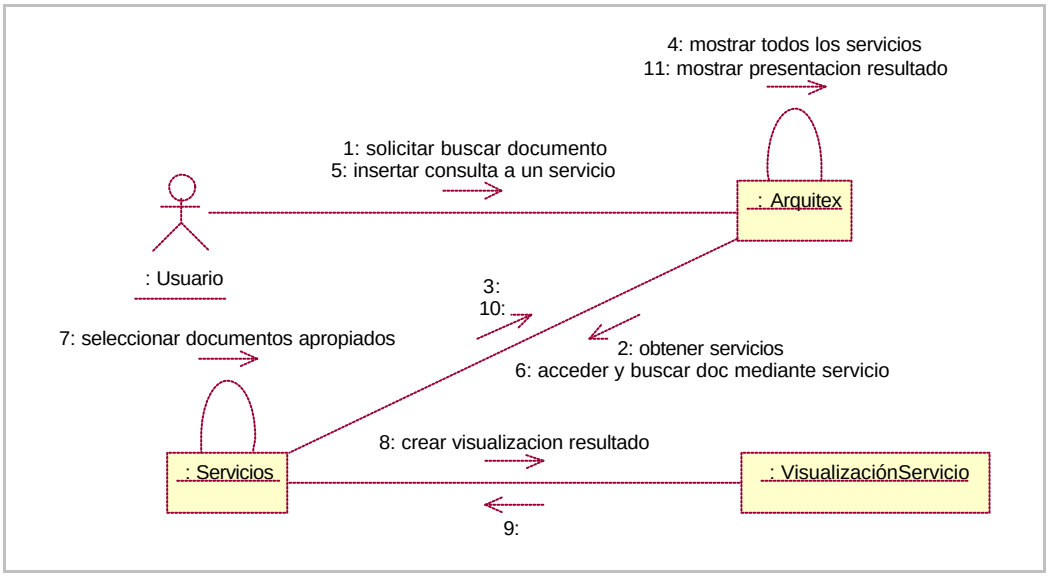


Figura 169: Análisis: Diagrama de colaboración: Consultar resultado de la búsqueda

La operación *"Acceder y realizar búsqueda de documento mediante servicio"* forma parte ya de la arquitectura. Para más información consulte el análisis y diseño de la arquitectura en la página 122.

4.2 Diseño de la aplicación Arquitex

En esta sección se detalla el diseño del sistema, cuyo objetivo es mostrar cómo se resuelve el problema planteado en la especificación de requisitos. Para ello se usan las recomendaciones de Conallen para modelar arquitecturas Web con UML que permite rentabilizar toda la gramática interna de UML para modelar aplicaciones con elementos específicos de la arquitectura de un entorno Web

4.2.1 Caso de uso: Gestionar Documentos de la colección

A continuación se procede a detallar cada uno de los escenarios que forman parte de este caso de uso.

Escenario: Añadir un documento a la colección

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

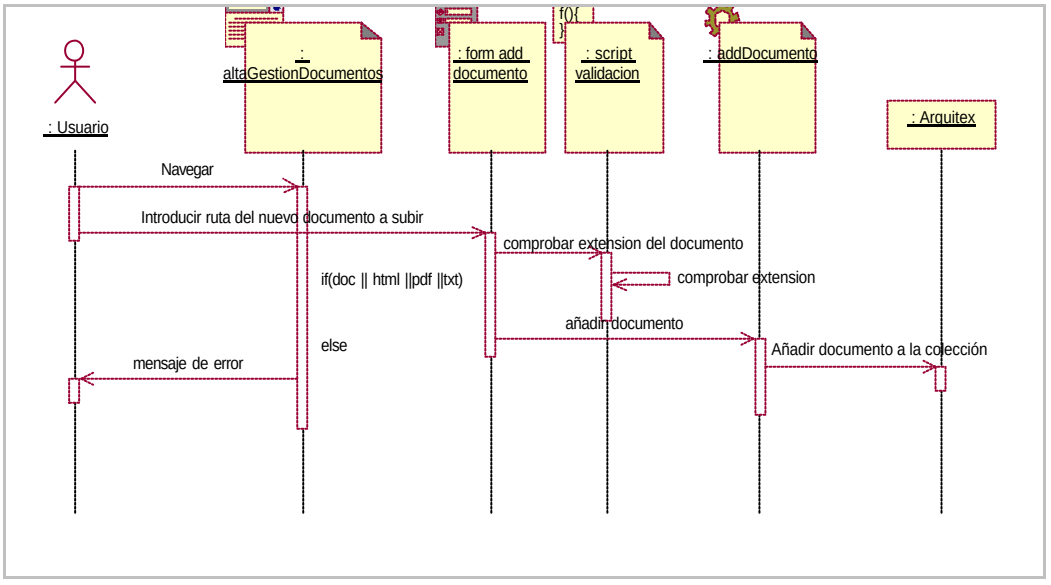


Figura 170: Diseño: Diagrama de secuencia: Añadir un documento a la colección

El diagrama de colaboración de este escenario:

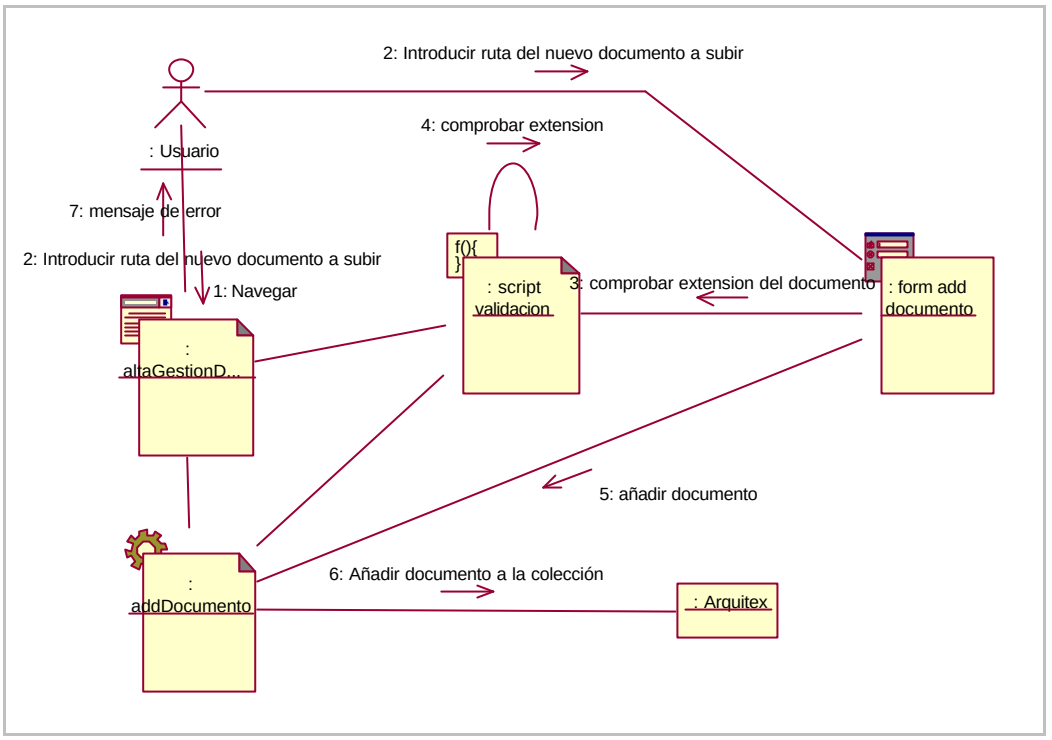


Figura 171: Diseño: Diagrama de colaboración: Añadir un documento a la colección

La operación "*Añadir documento a la colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

El diagrama de clases parcial es el siguiente:

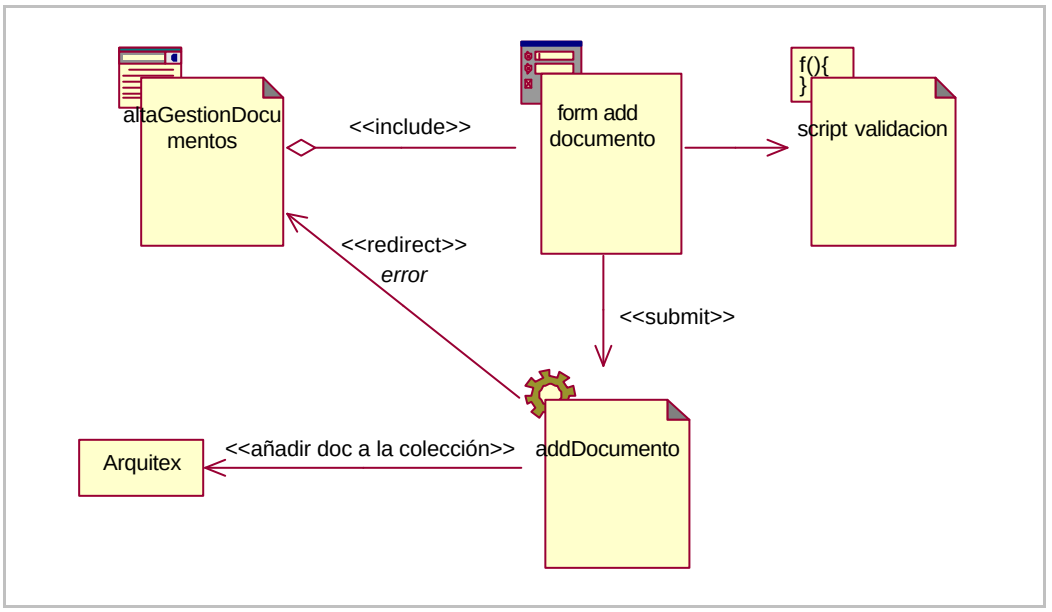


Figura 172: Diseño: Diagrama de clases parcial: Añadir un documento a la colección

Escenario: Eliminar un documento de la colección

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

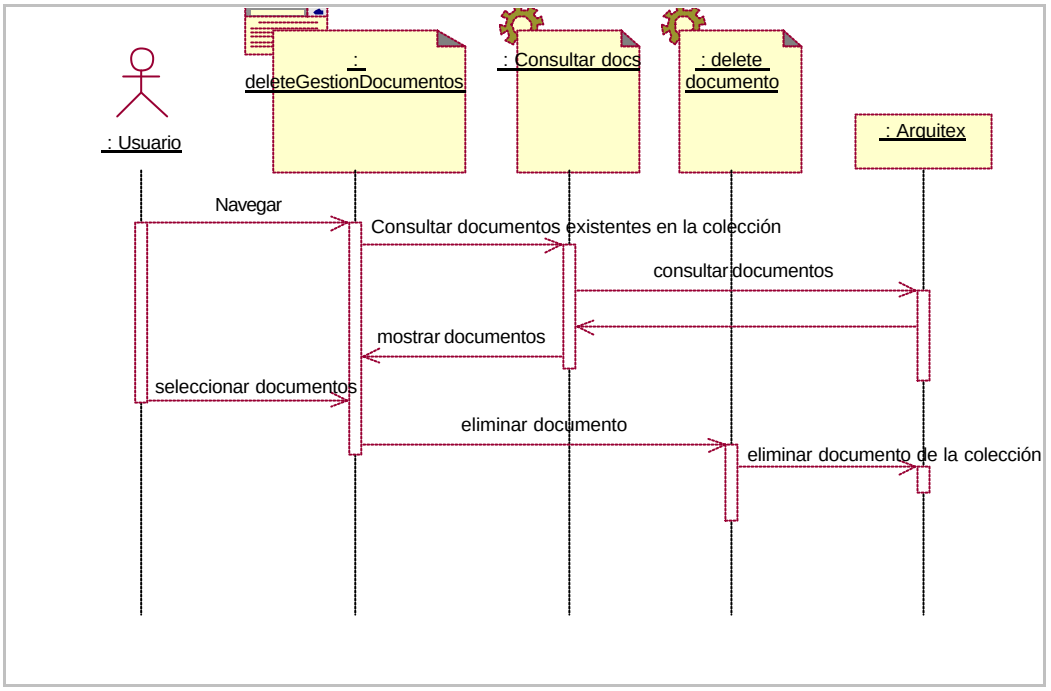


Figura 173: Diseño: Diagrama de secuencia: Eliminar un documento de la colección

El diagrama de colaboración de este escenario:

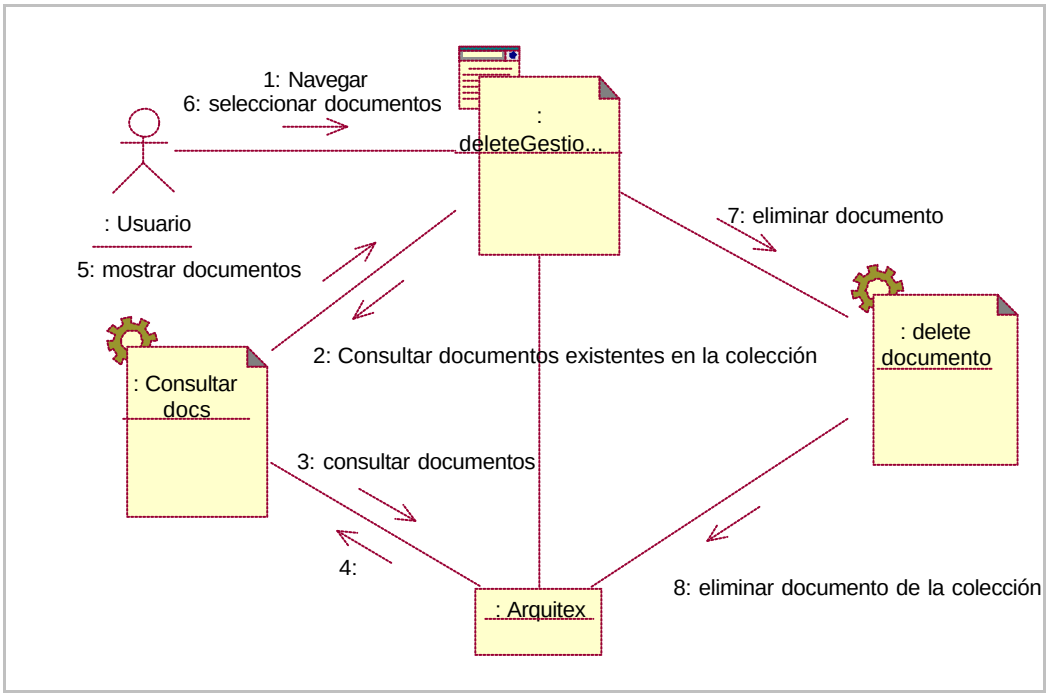


Figura 174: Diseño: Diagrama de colaboración: Eliminar un documento de la colección

La operación "*Eliminar documento a la colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

El diagrama de clases parcial es el siguiente:

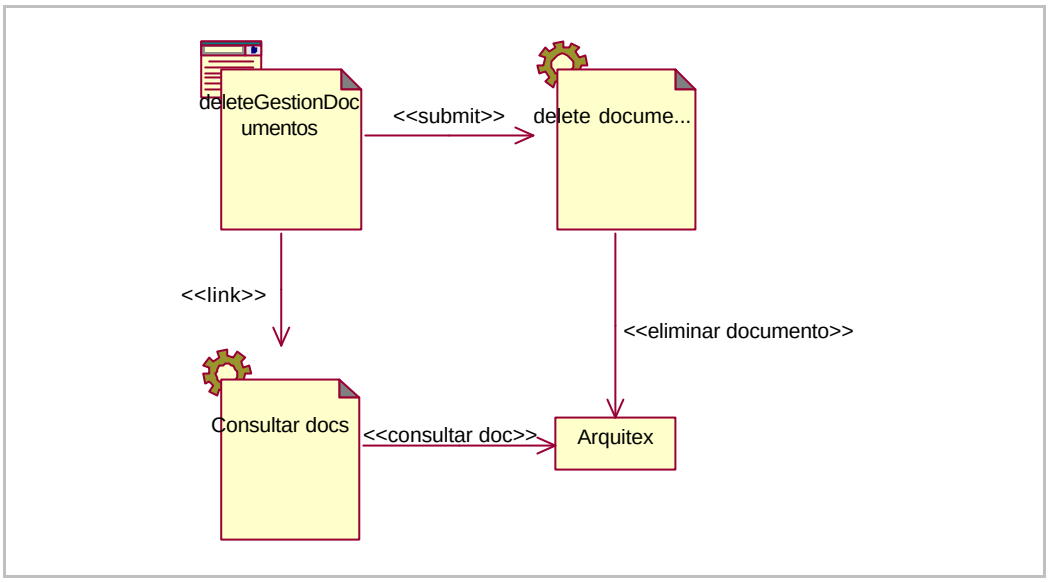


Figura 175: Diseño: Diagrama de clases parcial: Eliminar un documento de la colección

4.2.2 Caso de uso: Consultar ayuda

A continuación se procede a detallar el escenario que conforma este caso de uso.

Escenario: Consultar ayuda

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

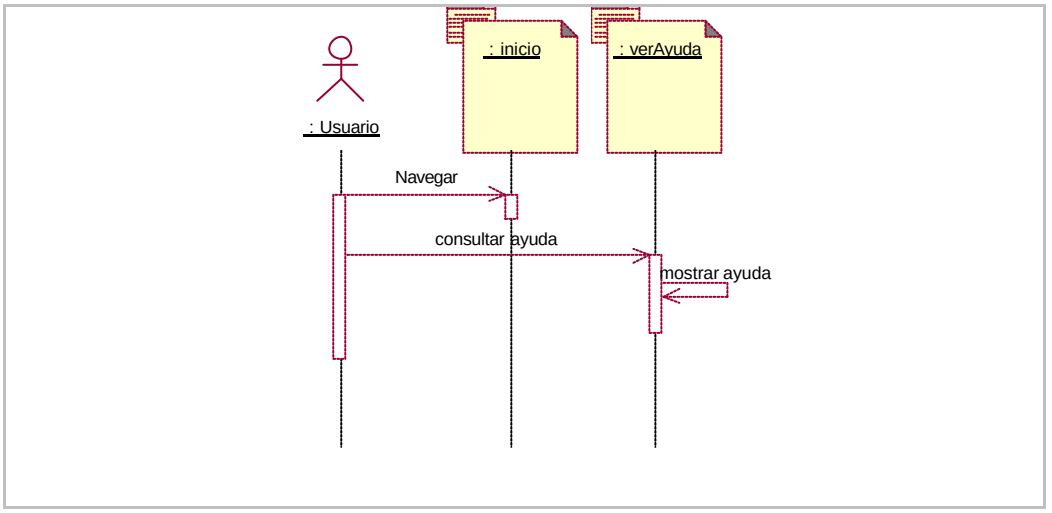


Figura 176: Diseño: Diagrama de secuencia: Consultar ayuda

El diagrama de colaboración de este escenario:

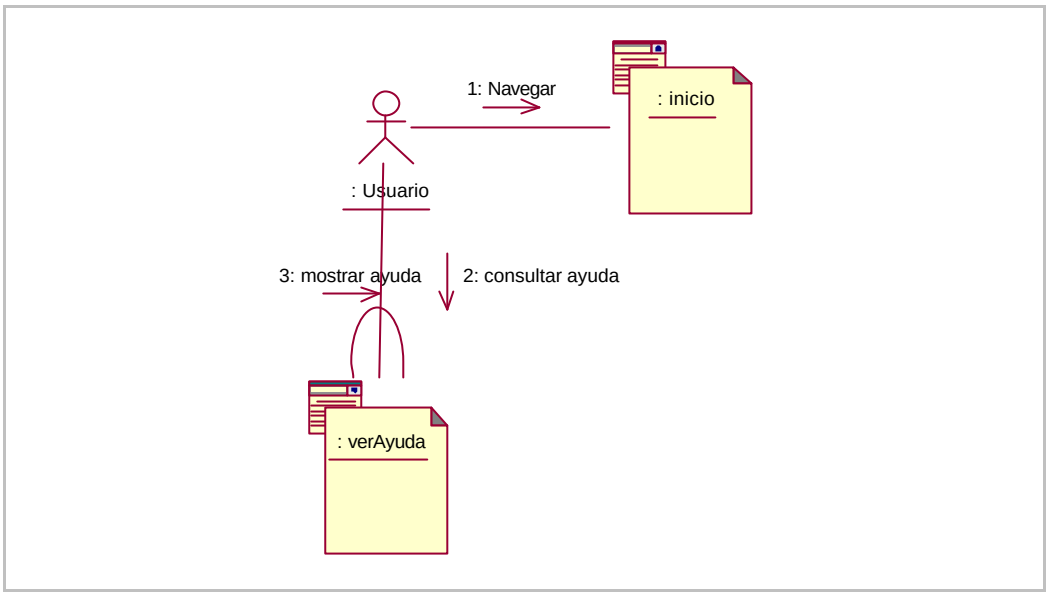


Figura 177: Diseño: Diagrama de colaboración: Consultar ayuda

El diagrama de clases parcial es el siguiente:

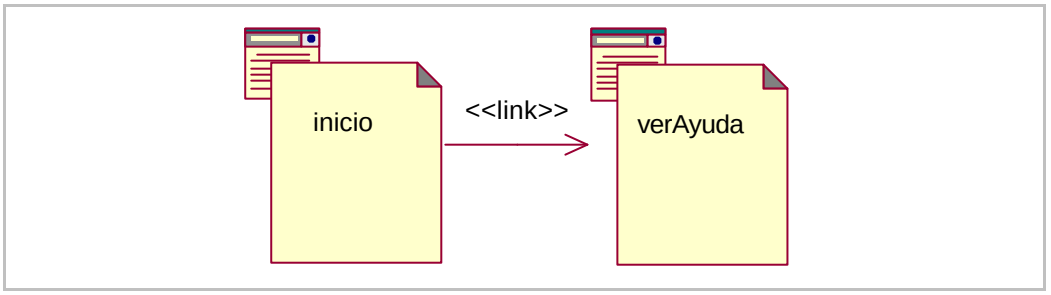


Figura 178: Diseño: Diagrama de clases parcial: Consultar ayuda

4.2.3 Caso de uso: Gestionar el fichero de configuración

A continuación se procede a detallar cada uno de los escenarios que satisfacen este de uso.

Escenario: Crear fichero de configuración

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

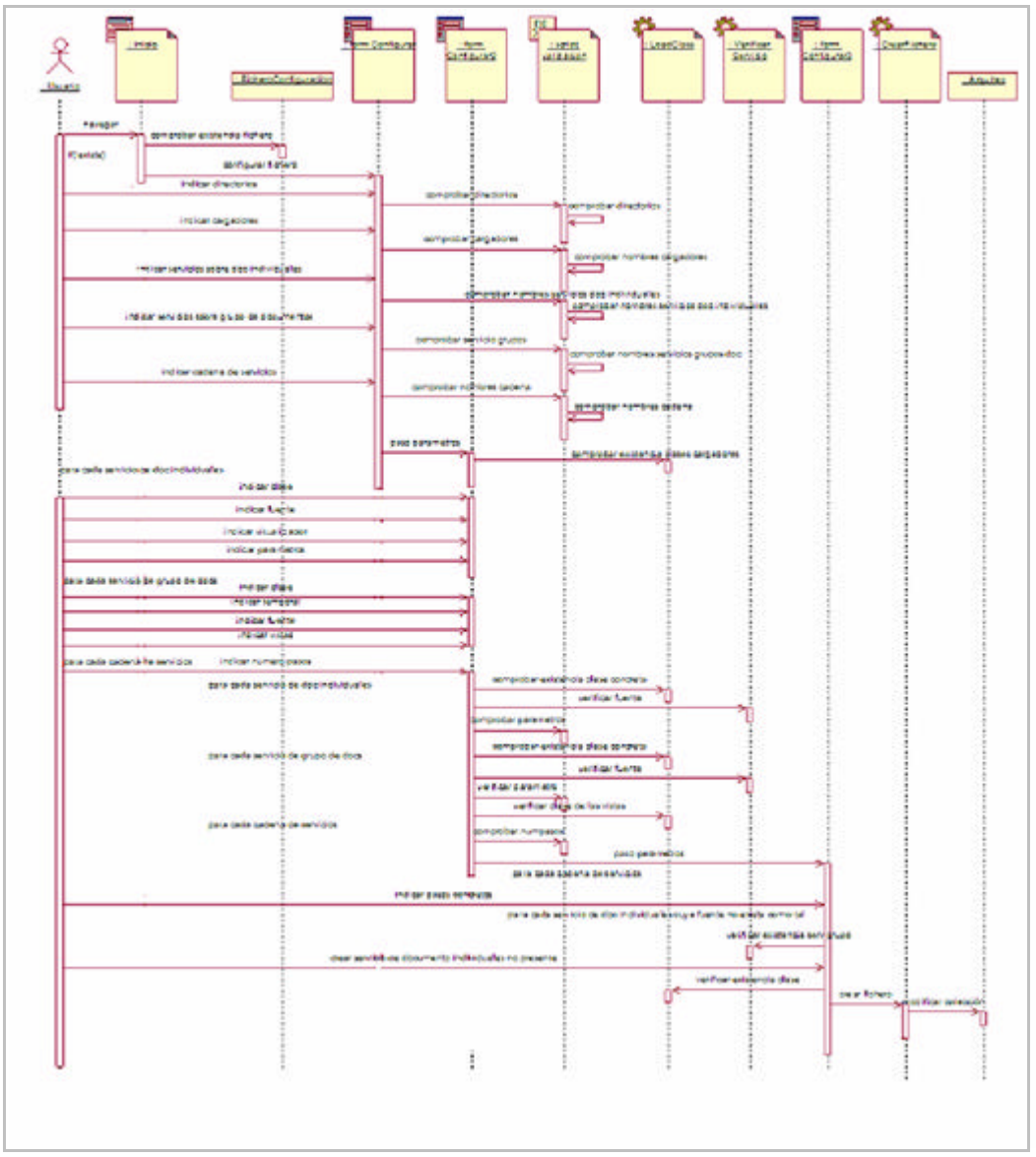


Figura 179: Diseño: Diagrama de secuencia: Crear fichero de configuración

El diagrama de colaboración de este escenario:

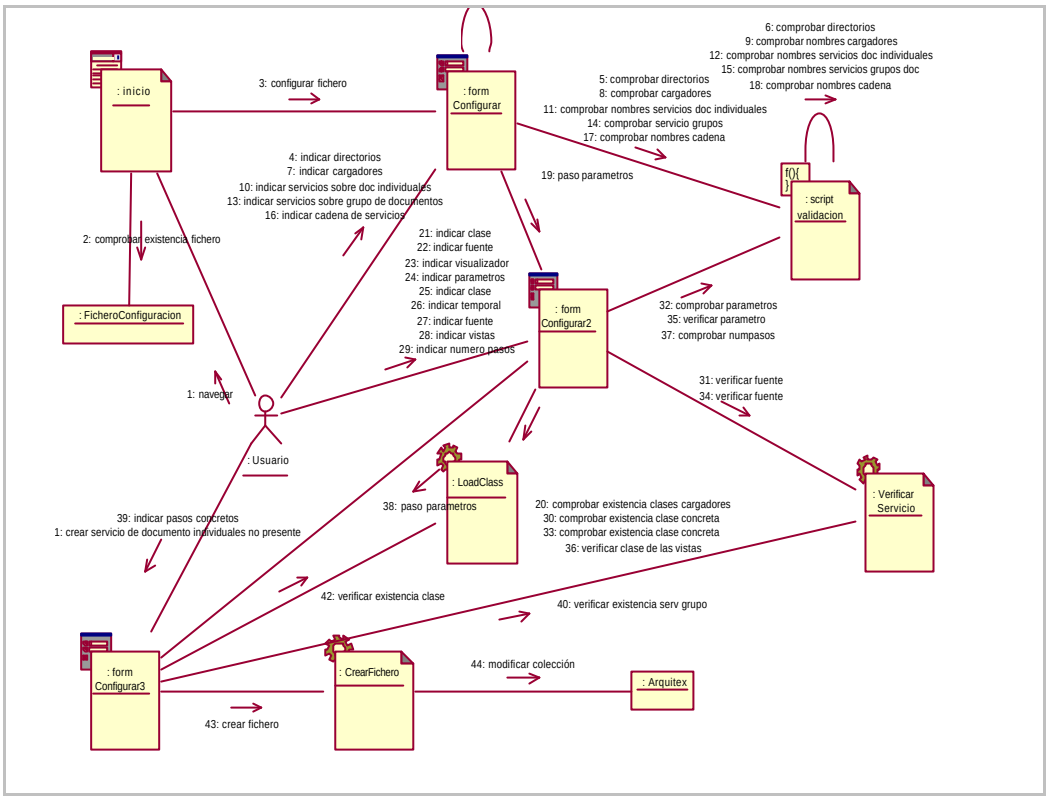


Figura 180: Diseño: Diagrama de colaboración: Crear fichero de configuración

La operación "*Inicializar la colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

El diagrama de clases parcial es el siguiente:

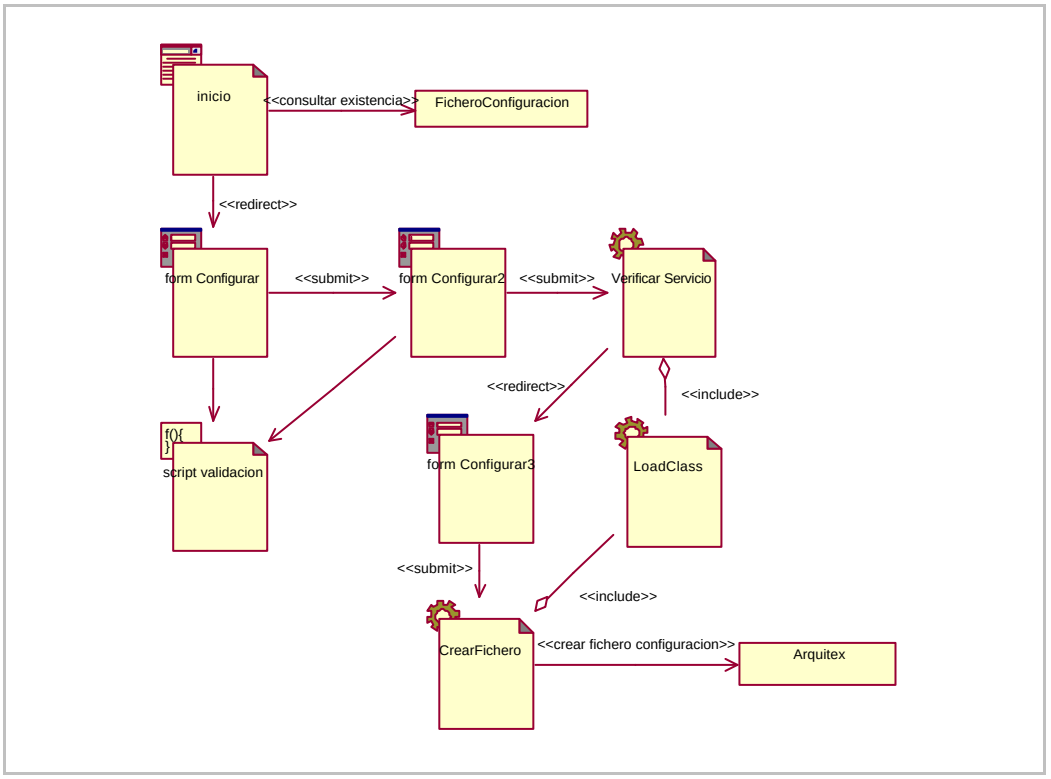


Figura 181: Diseño: Diagrama de clases parcial: Crear fichero de configuración

Escenario: Modificar el fichero de configuración

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

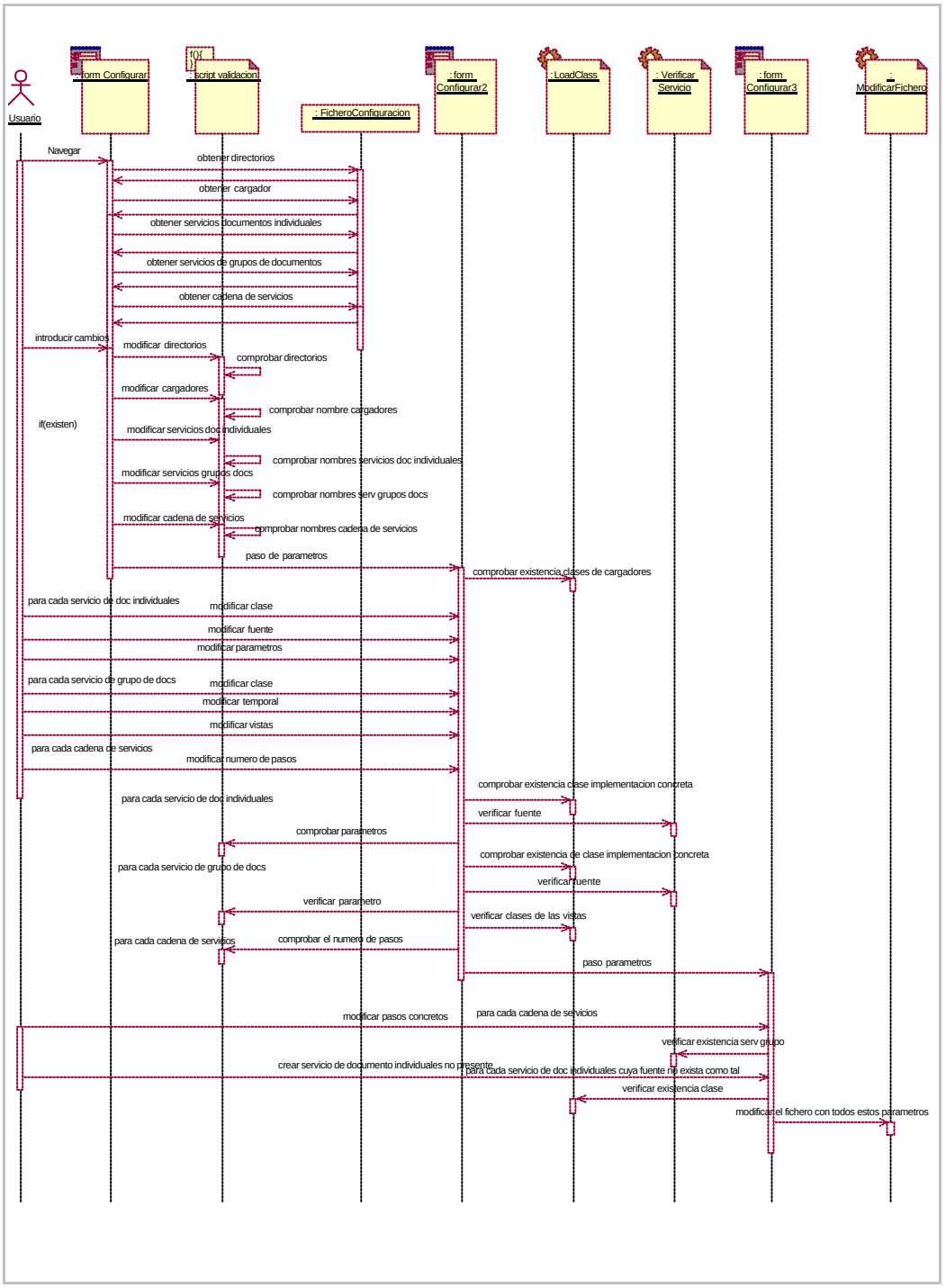


Figura 182: Diseño: Diagrama de secuencia: Modificar fichero de configuración

El diagrama de colaboración de este escenario:

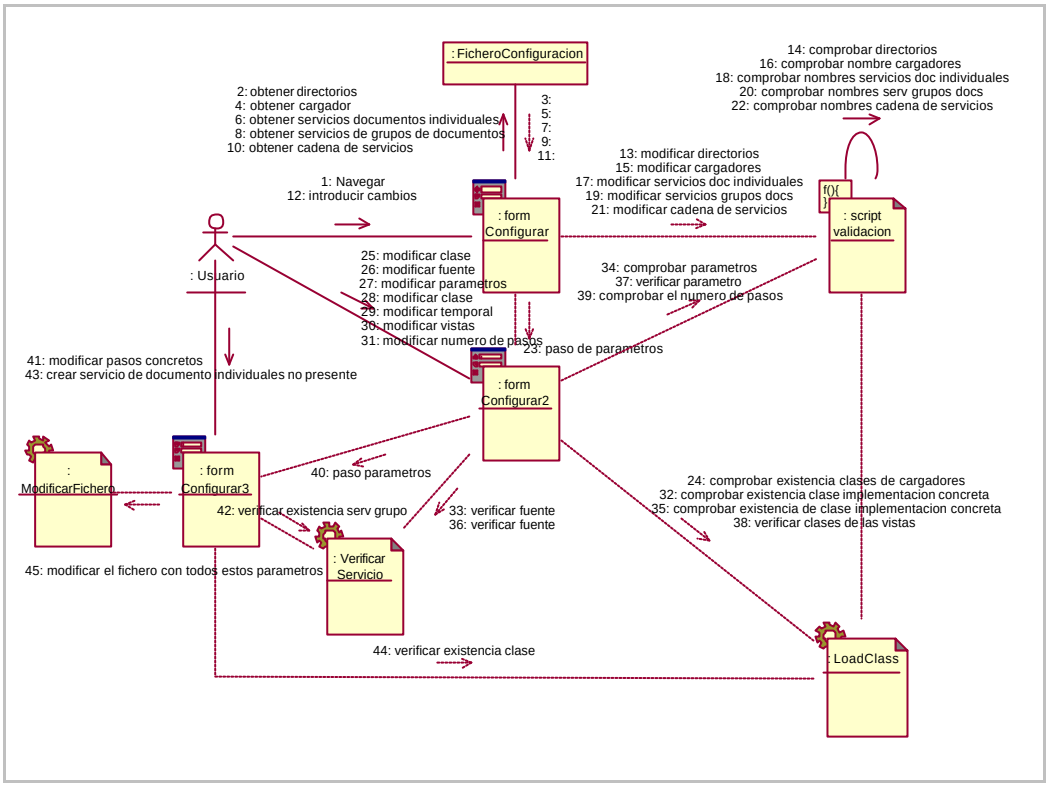


Figura 183: Diseño: Diagrama de colaboración: Modificar fichero de configuración

La operación "*Modificar colección*" forma parte ya de la arquitectura. Para más información consulte la página 106.

El diagrama de clases parcial es el siguiente:

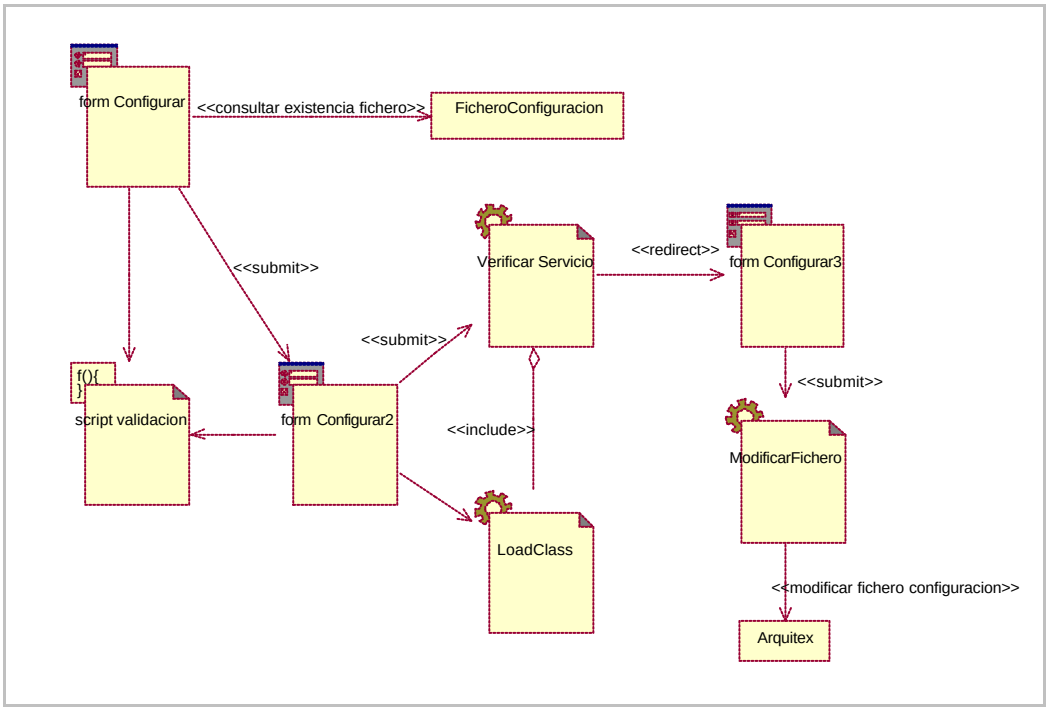


Figura 184: Diseño: Diagrama de clases parcial: Modificar fichero de configuración

4.2.4 Caso de uso: Gestionar las presentaciones

A continuación se procede a detallar cada uno de los escenarios que satisfacen este de uso.

Escenario: Consultar documento base

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

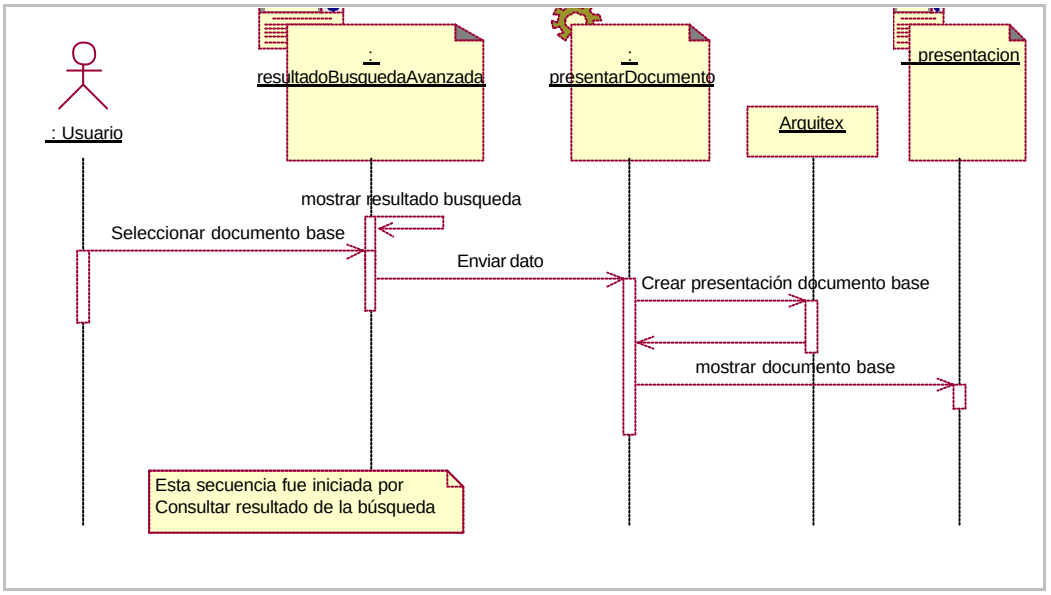


Figura 185: Diseño: Diagrama de secuencia: Consultar documento base

El diagrama de colaboración de este escenario:

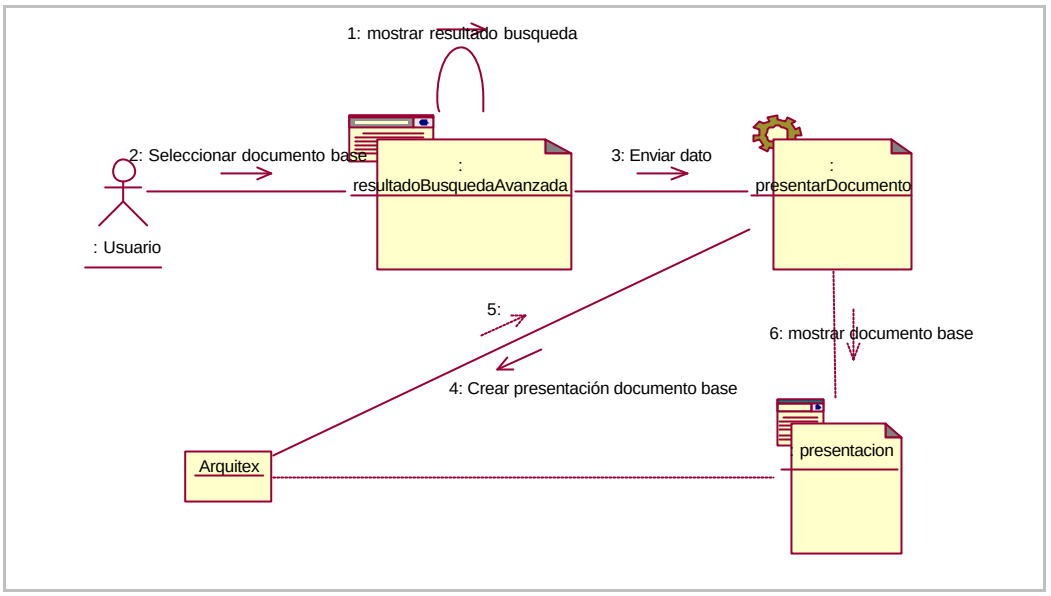


Figura 186: Diseño: Diagrama de colaboración: Consultar documento base

La operación "*Crear presentación documento base*" forma parte ya de la arquitectura. Para más información consulte el análisis y diseño de la arquitectura en la página 129.

El diagrama de clases parcial es el siguiente:

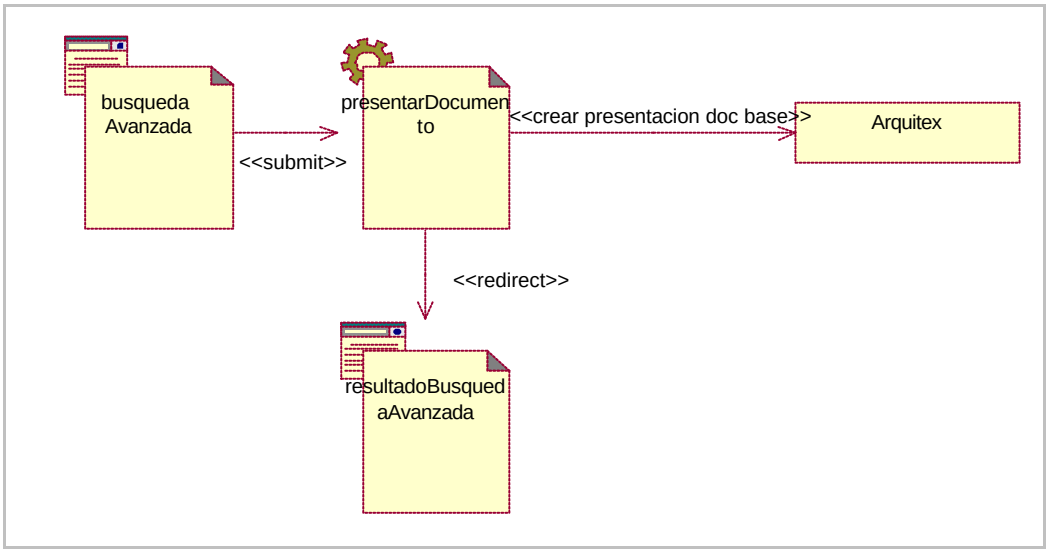


Figura 187: Diseño: Diagrama de clases parcial: Consultar documento base

Escenario: Consultar documento decorado

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

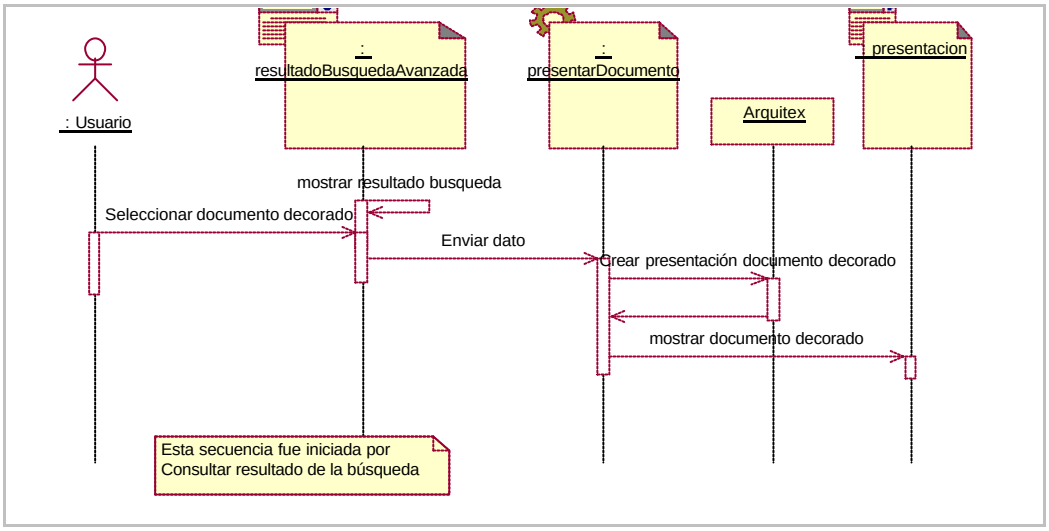


Figura 188: Diseño: Diagrama de secuencia: Consultar documento decorado

El diagrama de colaboración de este escenario:

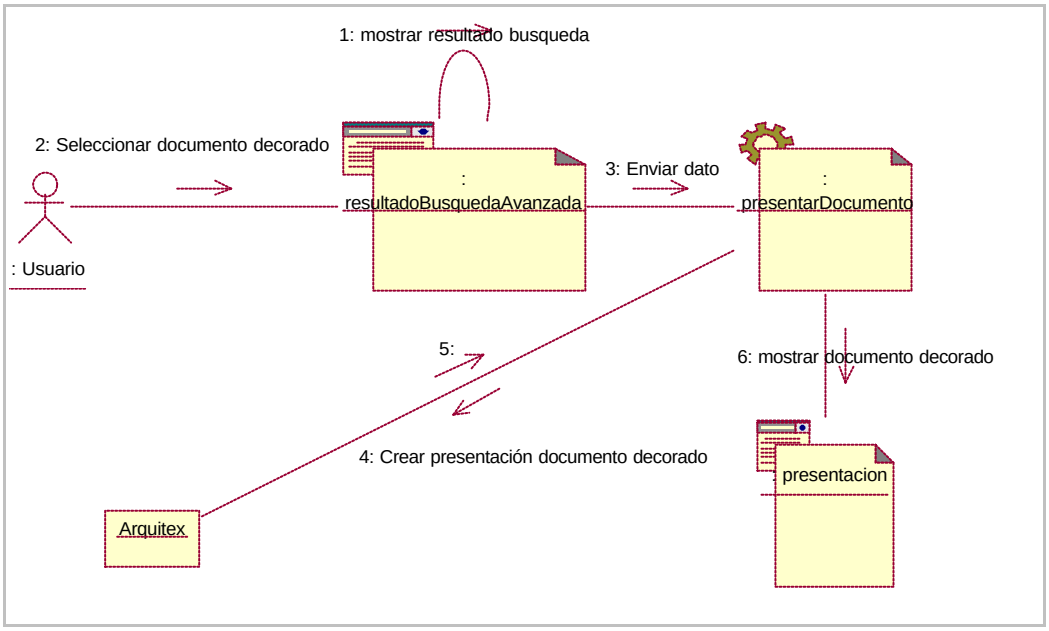


Figura 189: Diseño: Diagrama de colaboración: Consultar documento decorado

La operación *"Crear presentación documento decorado"* forma parte ya de la arquitectura. Para más información consulte el análisis y diseño de la arquitectura en la página 129.

El diagrama de clases parcial es el siguiente:

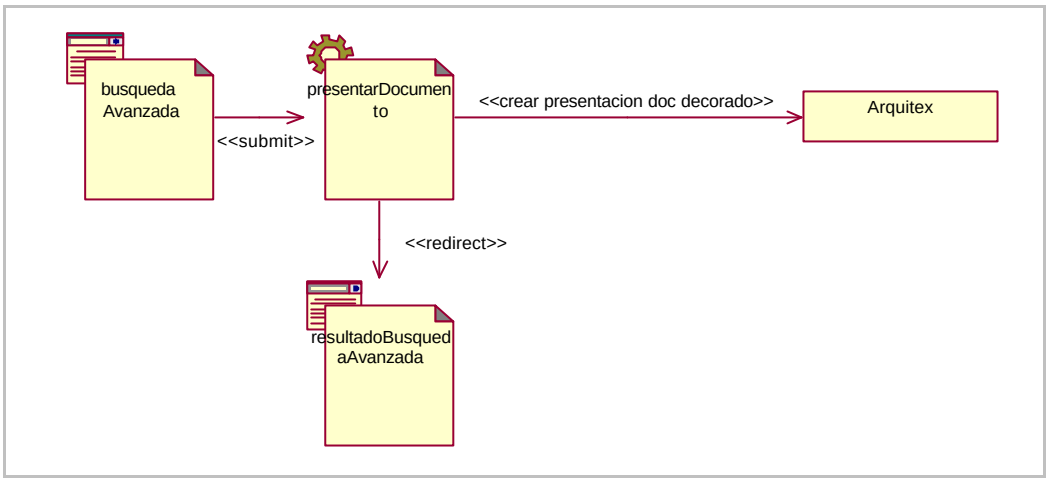


Figura 190: Diseño: Diagrama de clases parcial: Consultar documento decorado

4.2.5 Caso de uso: Gestionar Búsquedas

A continuación se procede a detallar cada uno de los escenarios que satisfacen este de uso.

Escenario: Consultar resultado de la búsqueda

El proceso llevado a cabo se puede ver detalladamente en el siguiente diagrama de secuencia.

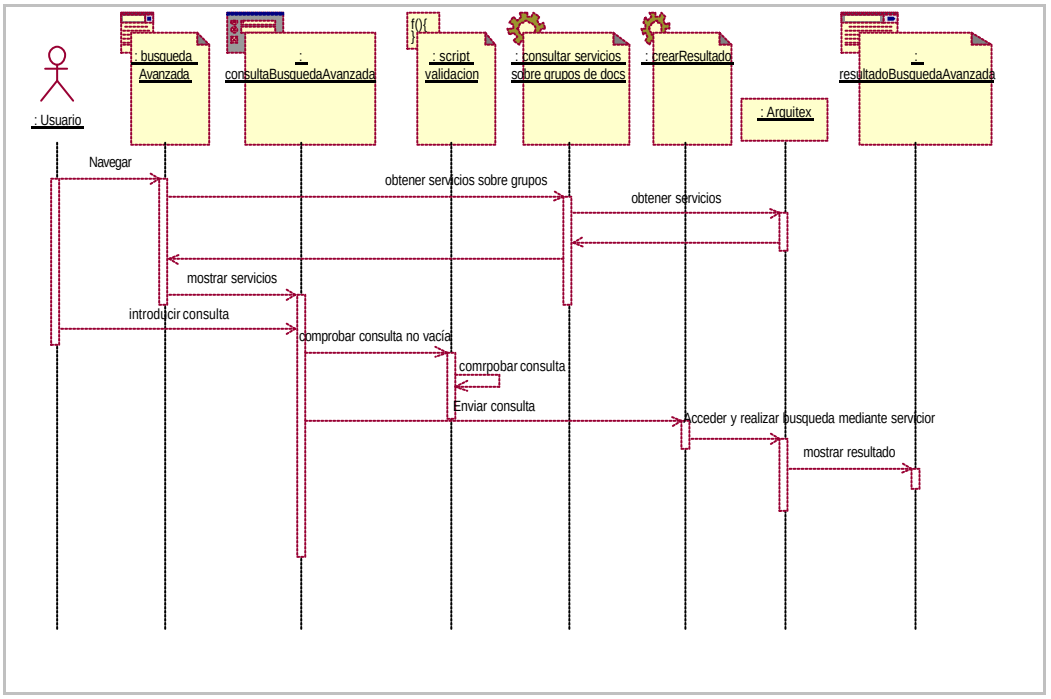


Figura 191: Diseño: Diagrama de secuencia: Consultar resultado de la búsqueda

El diagrama de colaboración de este escenario:

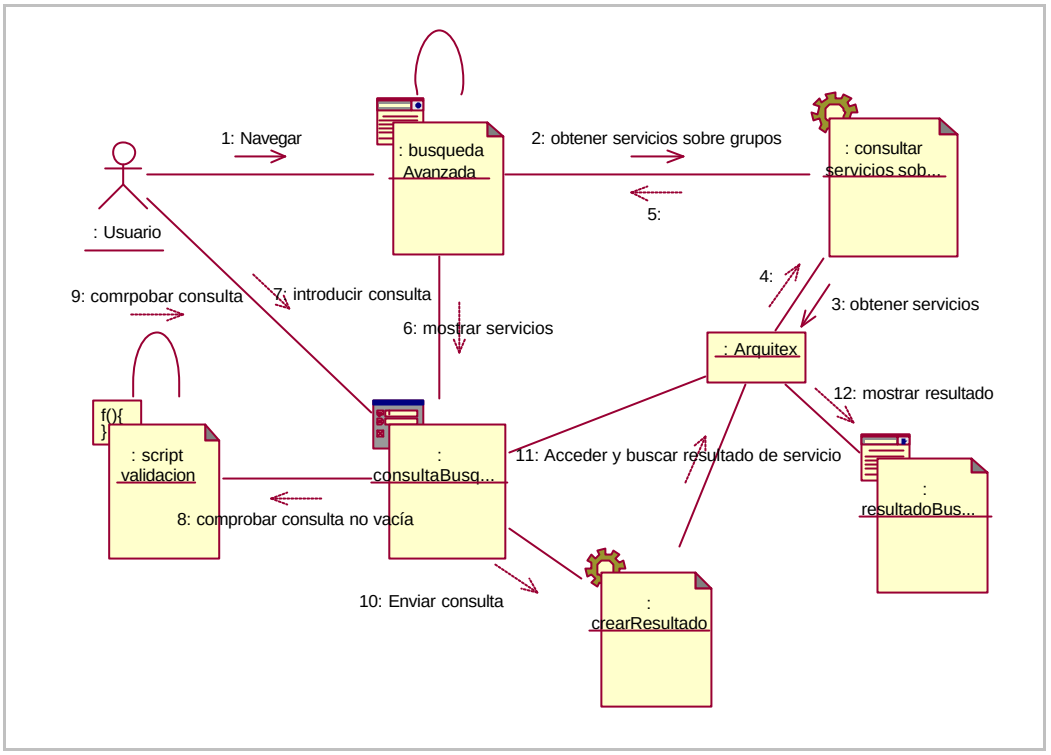


Figura 192: Diseño: Diagrama de colaboración: Consultar resultado de la búsqueda

La operación *"Acceder y realizar búsqueda de documento mediante servicio"* forma parte ya de la arquitectura. Para más información consulte el análisis y diseño de la arquitectura en la página 122.

El diagrama de clases parcial es el siguiente:

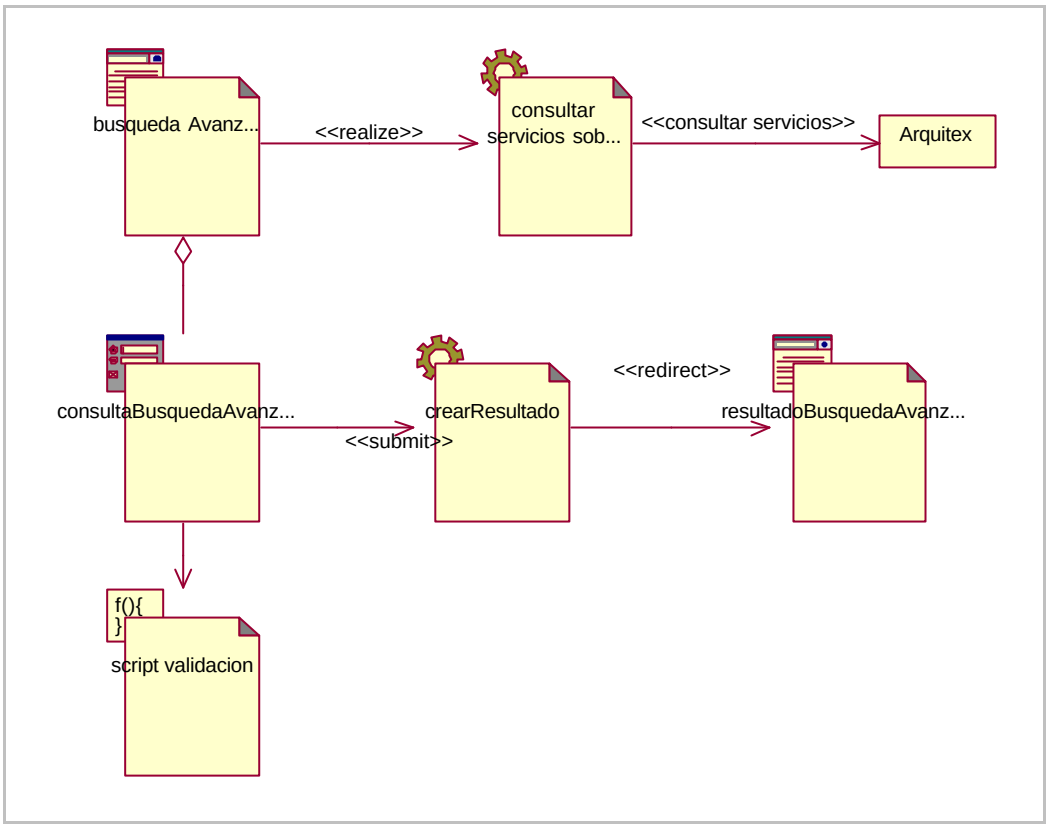


Figura 193: Diseño: Diagrama de clases parcial: Consultar resultado de la búsqueda

5. Implementación y pruebas

A continuación se van a detallar ciertos aspectos del sistema que es conveniente aclarar.

5.1 Formato del fichero de configuración

Las propiedades de las que está compuesto el fichero de configuración tienen el siguiente significado:

// Directorios donde almacenar los documentos

directorio.default = C:/Archivosdeprograma/ApacheGroup/Tomcat 5.0/webapps/arquitex ó /home/mila/tomcat/webapps/arquitex

- Propiedad que indica la ruta absoluta donde se van a alojar la estructura de directorios generada por la propia arquitectura.

directorioXML.default = /c/XML/

- Propiedad que indica la ruta relativa del directorio donde se van a alojar los documentos base de la colección. Una vez concatenada a la que posee *directorio.default*, obtenemos la ruta absoluta del directorio de documentos base.

directorioDeco.default = /c/decoraciones/

- Propiedad que indica la ruta relativa del directorio donde se van a alojar los documentos decorados de la colección. Una vez concatenada a la que posee *directorio.default*, obtenemos la ruta absoluta del directorio de documentos decorados.

directorioServicios.default = /c/servicios/

- Propiedad que indica la ruta relativa del directorio donde se van a alojar los servicios sobre grupos de documentos de la colección. Una vez concatenada a la que posee *directorio.default*, obtenemos la ruta absoluta del directorio de servicios sobre grupos de documentos.

directorioVisual.default = /c/visual/

- Propiedad que indica la ruta relativa del directorio donde se van a alojar los servicios de presentación de los documentos base y decorados de la colección. Una vez concatenada a la que posee *directorio.default*, obtenemos la ruta absoluta del directorio de servicios de presentaciones de documentos.

// Clases cargadoras de información

cargador.html.clase = ConvertHTML2XML

- Propiedad que indica la dase que va a ser cargada en tiempo de ejecución cuando los documentos que se le pasen sean de tipo HTML. La clase *ConvertHTML2XML* permite extraer la información de un determinado tipo de HTML con un formato adecuado (ver apartado al respecto).

cargador.doc.clase =

- Propiedad que indica la clase que va a ser cargada en tiempo de ejecución cuando los documentos que se le pasen sean de tipo DOC. No se encuentra implementado ningún cargador para este tipo de documentos en este proyecto.

cargador.pdf.clase =

- Propiedad que indica la clase que va a ser cargada en tiempo de ejecución cuando los documentos que se le pasen sean de tipo PDF. No se encuentra implementado ningún cargador para este tipo de documentos en este proyecto.

cargador.txt.clase =ConvertTXT2XML

- Propiedad que indica la clase que va a ser cargada en tiempo de ejecución cuando los documentos que se le pasen sean de tipo TXT. La clase *ConvertTXT2XML* permite extraer la información de un documento con texto plano.

// Lista de servicios sobre documentos individuales

resumidores = classifier4j

- Propiedad que permite indicar los nombres de los resumidores que va a hacer uso la arquitectura en el momento de ejecutarse. Todos los resumidores estarán separados por comas. En caso contrario, se considerará como único. En este caso va a existir un resumidor con nombre *classifier4j*.

traductores = google_en,google_fr

- Propiedad que permite indicar los nombres de los traductores que va a hacer uso la arquitectura en el momento de ejecutarse. Todos los traductores estarán separados por comas. En caso contrario, se considerará como único. En este caso van a existir dos traductores: uno con nombre *google_en* y otro con nombre *google_fr*.

clasificadores = weka

- Propiedad que permite indicar los nombres de los clasificadores que va a hacer uso la arquitectura en el momento de ejecutarse. Todos los clasificadores estarán separados por comas. En caso contrario, se considerará como único. En este caso va a existir un clasificador con nombre *weka*.

extractores = erial

- Propiedad que permite indicar los nombres de los extractores de palabras clave que va a tener que hacer uso la arquitectura en el momento de ejecutarse. Todos los extractores estarán separados por comas. En caso contrario, se considerará como único. En este caso va a existir un extractor con nombre *erial*.

// Lista de servicios sobre grupos de documentos

indexadores = lucene_es,lucene_classifier4j

- Propiedad que permite indicar los nombres de los indexadores que va a tener que hacer uso la arquitectura en el momento de ejecutarse. Todos los indexadores estarán separados por comas. En caso contrario, se considerará como único. En este caso van a existir dos indexadores, uno con nombre *lucene_es* y otro con nombre *lucene_classifier4j*.

clusterings = weka_cluster

- Propiedad que permite indicar los nombres de los clusters que va a tener que hacer uso la arquitectura en el momento de ejecutarse. Todos los clusters estarán separados por comas. En caso contrario, se considerará como único. En este caso va a existir un clustering con nombre *weka_cluster*.

// Lista de cadena de servicios

cadena_de_servicios = IndexadorIndexador

- Propiedad que permite indicar los nombres de las cadenas de servicios que se van a considerar y que va a hacer uso la arquitectura en el momento de ejecutarse. Todas las cadenas de servicios estarán separadas por comas. En caso contrario, se considerará como único. En este caso va a existir una cadena con nombre *IndexadorIndexador* que se detalla más abajo.

Antes de seguir es necesario indicar que no pueden existir nombres repetidos ni en los servicios sobre documentos individuales, ni en servicios sobre grupos de documentos ni en las cadenas de servicios.

DocumentoBase.visualizador = VisualizacionHTMLNormal

- Propiedad que indica la clase que va a ser empleada para la presentación y visualización de los documentos base. Ésta se llamará en tiempo de ejecución cuando se creen las visualizaciones de los documentos base. En este caso, la clase *VisualizacionHMTLDocNormal* hace una llamada en todos sus métodos de una clase llamada *VisualizacionHTML* que está incluida por defecto en la arquitectura.

// A continuación se describen cada uno de los servicios sobre documentos individuales que se indicó más arriba. Cabe destacar la necesidad de que todas propiedades estén cubiertas

// Resumidores

classifier4j.clase = ResumidorClassifier4j

classifier4j.fuente = DocumentoBase

classifier4j.visualizador = VisualizacionHTMLResumen

classifier4j.lineas = 4

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad resumidores. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase indica la clase que va a implementar ese resumidor. En este caso va a ser la clase *ResumidorClassifier4j*.
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos base.

- El atributo visualizador indica la clase que se va a usar para visualizar la decoración que se genera mediante la clase *ResumidorClassifier4j*. Esta clase no hace uso del visualizador que trae la arquitectura por defecto, sino que fue implementada para mostrar como se haría.
- Los atributos que siguen los tres anteriores son parámetros propios de cada una de las implementaciones que se haga. En este caso, el resumidor implementado usa la API Classifier4j cuyo parámetro es justamente el número de líneas en las que se quiere que realice el resumen.

classifier5j.clase = ResumidorClassifier4j
classifier5j.fuente = classifier4j
classifier5j.visualizador = VisualizacionHTMLResumen
classifier5j.lineas = 1

- Idem que el anterior pero con la diferencia de la fuente sobre la que se va a crear esta decoración es ya una decoración, es decir, se realizará un resumen sobre una visión que no es otra que la explicada anteriormente.

// Traductores

google_en.clase = TraductorGoogle
google_en.fuente = DocumentoBase
google_en.visualizador = VisualizacionHTMLNormal
google_en.origen = es
google_en.destino = en

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad traductores. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase indica la clase que va a implementar ese traductor. En este caso va a ser la clase *TraductorGoogle*.
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos base.
  - El atributo visualizador indica la clase que se va a usar para visualizar la decoración que se genera mediante la clase *TraductorGoogle*.
  - Los atributos que siguen los tres anteriores son parámetros propios de cada una de las implementaciones que se haga. En este caso, el traductor implementado usa la herramienta Google cuyos parámetros son justamente el origen de los documentos, es decir el idioma inicial del documento sobre el que se quiere obtener una visión, y el destino de los documentos, es decir, el idioma en el cual queremos que no sea traducida la fuente.

google_fr.clase = TraductorGoogle
google_fr.fuente = google_en
google_fr.visualizador = VisualizacionHTMLNormal
google_fr.origen = en
google_fr.destino = fr

- Idem que el anterior pero con la diferencia de la fuente sobre la que se va a crear esta decoración es ya una decoración, es decir, se realizará una traducción sobre una visión que no es otra que la explicada anteriormente. Cabe destacar que la herramienta Google no posee una traducción directa del idioma español al idioma francés por lo que es necesario realizar un paso intermedio que no es otro que la traducción del español al inglés y del inglés al francés.

// Extractor

erial.clase = ExtractorErial
erial.fuente = DocumentoBase
erial.dirScriptErial = /home/mila/Erial
erial.visualizador = VisualizacionHTMLNormal

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad extractor. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase indica la clase que va a implementar ese extractor. En este caso va a ser la clase *ExtractorErial*.
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos base.
  - El atributo visualizador indica la clase que se va a usar para visualizar la decoración que se genera mediante la clase *ExtractorErial*.
  - Los atributos que siguen los tres anteriores es un parámetro propio de cada una de las implementaciones que se haga. En este caso, el extractor implementado usa la herramienta Erial cuyo parámetro es justamente la dirección del script de Erial.

//Clasificador

weka.clase = ClasificadorWEKASimple
weka.fuente = DocumentoBase
weka.visualizador = VisualizacionHTMLNormal
weka.palabras = pal.txt
weka.entrenamiento = vec2.arff
weka.claseClasificadora = weka.classifiers.trees.J48

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad clasificador. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase indica la clase que va a implementar ese clasificador. En este caso va a ser la clase *ClasificadorWekaSimple*.
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos base.
  - El atributo visualizador indica la clase que se va a usar para visualizar la decoración que se genera mediante la clase *ClasificadorWekaSimple*.

- Los atributos que siguen los tres anteriores es un parámetro propio de cada una de las implementaciones que se haga. En este caso, el clasificador implementado usa la herramienta Weka cuyos parámetros son justamente el fichero que contiene las palabras clave, el fichero de entrenamiento y la clase que va a realizar la clasificación.

//….Idem para el resto de servicio sobre documentos individuales.

//Clustering

weka_cluster.clase = ClusterWEKASimple
weka_cluster.fuente = DocumentoBase
weka_cluster.temporal = falso
weka_cluster.visualizador = VisualizacionServHTMLConcreto
weka_cluster.vistaLista = classifier4j
weka_cluster.palabras = pal.txt
weka_cluster.opciones =-N 5
weka_cluster.claseCluster = weka.clusterers.SimpleKMeans

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad clusterings. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase indica la clase que va a implementar ese cluster. En este caso va a ser la clase *ClusterWEKASimple.*
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos base.
  - El atributo temporal indica si va a existir continuamente una estructura relacionada con el cluster o si se va a generar cada vez que se solicite. En este caso, indica que solo se genera una vez y ésta se mantiene persistente durante la ejecución.
  - El atributo visualizador indica la clase que se va a usar para visualizar el resultado de la consulta mediante la clase *ClusterWEKASimple*.
  - El atributo vistaLista indica cual va ser el contenido de los listados de los documentos tras realizar una consulta mediante cluster. En este caso, se mostrará un breve resumen generado por classifier4j
  - El atributo que sigue los cinco anteriores son parámetros propios de cada una de las implementaciones que se haga. En este caso, el clustering Weka posee por un lado el documento de entrenamiento, se le indica parámetros propios del cluster como es –N 5 que indica que se devolverán 5 grupos, y la clase encargada de realizar el cluster como es weka.clusterers.SimpleKMeans.

//Indexadores

lucene_en.clase = IndexadorLucene
lucene_en.fuente =google_en
lucene_en.temporal = falso
lucene_en.visualizador = VisualizacionServHTMLConcreto
lucene_en.vistaLista =DocumentoBase

lucene_en.fuenteConsulta=google_en

lucene_en.claseAnalizadora = StopAnalyzer

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad indexadores. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase indica la clase que va a implementar ese cluster. En este caso va a ser la clase *IndexadorLucene*.
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos generados por google_en.
  - El atributo temporal indica si va a existir continuamente una estructura relacionada con el indexador o si es necesario crear el índice de cada vez. En este caso, indica que solo se genera una vez y ésta se mantiene persistente durante la ejecución.
  - El atributo visualizador indica la clase que se va a usar para visualizar el resultado de la consulta mediante la clase *IndexadorLucene*.
  - El atributo vistaLista indica cual va ser el contenido de los listados de los documentos tras realizar una consulta mediante indexador
  - El atributo fuenteConsulta indica cual es la transformación que se le debe de realizar a la consulta pasada por el usuario para procurar homogeneizarla con el resultado, es decir, en este caso la consulta va a sufrir la misma transformación que la fuente del indexador. En este caso partimos de documentos en español y obtenemos su traducción. Esa traducción es lo que vamos a indexar. En el momento de hacer la consulta en español, el sistema transformará esa consulta según la fuenteConsulta (no es otra que traducir la consulta) para obtener los resultados.
  - El atributo que sigue los seis anteriores son parámetros propios de cada una de las implementaciones que se haga. En este caso, el indexador Lucene posee la ventaja de poder escoger cual va a ser la clase que realice el análisis de los documentos para eliminar la lista de parada.

lucene_classifier4j.clase = IndexadorLucene

lucene_classifier4j.fuente = classifier4j

lucene_classifier4j.temporal = falso

lucene_classifier4j.visualizador = VisualizacionServHTMLConcreto

lucene_classifier4j.vistaLista=classifier4j

lucene_classifier4j.fuenteConsulta=classifier4j

lucene_classifier4j.claseAnalizadora = SimpleAnalyzer

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad indexadores. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo clase: idem anterior.
  - El atributo fuente indica cuales son los documentos sobre los que se va a realizar la visión. En este caso se parte de los documentos generados por classifier4j.
  - El atributo temporal: idem anterior.
  - El atributo visualizador: idem anterior.

- El atributo vistaLista indica cual va ser el contenido de los listados de los documentos tras realizar una consulta mediante indexador. En este caso se visualizará un pequeño resumen.
- El atributo fuenteConsulta indica cual es la transformación que se le debe de realizar a la consulta pasada por el usuario para procurar homogeneizarla con el resultado, es decir, en este caso la consulta va a sufrir la misma transformación que la fuente del indexador. En este caso partimos de documentos en español y obtenemos su resumen mediante classifier4j. Ese resumen es lo que vamos a indexar.

// Idem para el resto de servicios sobre grupos de documentos…

//Cadenas de servicios

IndexadorIndexador.numpasos = 2
IndexadorIndexador.1 = lucene_classifier4j
IndexadorIndexador.2 = weka_cluster

- La primera parte del nombre de la propiedad procede de cada una de los elementos de los que se compone la propiedad cadena_de_servicios. Se va a proceder a indicar cada una de un modo detallado:
  - El atributo numpasos indica cuantos pasos será necesario realizar para obtener el resultado de la cadena de servicio.
  - Para cada uno de los pasos es necesario indicar que servicio sobre grupos de documentos va a usar. Existe una restricción: si se va a hacer uso de un método de agrupamiento, éste solo puede ser el último paso. El motivo viene detallado en la parte de Problemas encontrados.

5.2 Formato de la DTD

A continuación se describe la DTD asociada a los ficheros XML que resultan de los documentos base que se generan.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- DTD para documentosTextuales -->

<!ELEMENT documentoTextual (hash,titulo,capitulo+)>
```

- Etiqueta que contiene la toda la información acerca del documento textual. Éste contendrá un identificador del documento, un titulo, uno o varios capitulos.

```
<!ATTLIST hash numero CDATA #REQUIRED>
```

- Etiqueta que contiene el identificador del documento original de tipo carácter y éste será necesario

```
<!ELEMENT titulo (#PCDATA)>
```

- Etiqueta que contiene el titulo del documento.

<!ELEMENT capitulo (tituloCap,parrafo+)>

- Etiqueta que contiene toda la información acerca de los capítulos. Éste estará formado por el titulo del capitulo y por uno o varios párrafos.

<!ELEMENT tituloCap (#PCDATA)>

- Etiqueta que contiene el titulo del capitulo

<!ELEMENT parrafo (texto?,subparrafo*,imagen?)>

- Etiqueta que contiene la información de los párrafos del capitulo. Éste estará formado por uno o cero textos, cero o varios subpárrafos y cero o una imagen.

<!ELEMENT texto (#PCDATA)>

- Etiqueta que contiene el texto asociado a un párrafo.

<!ELEMENT imagen (#PCDATA)>

- Etiqueta que contiene la ruta donde se almacena la imagen asociada a un parrafo.

<!ELEMENT subparrafo (subtitulo,subtexto*,subimagen*)>

- Etiqueta que contiene toda la información acerca de los subpárrafos. Éste contiene un titulo asociado al subpárrafo, puede contener o no subtextos y subimagenes.

<!ELEMENT subtitulo (#PCDATA)>

- Etiqueta que contiene el titulo asociado al subpárrafo.

<!ELEMENT subtexto (#PCDATA)>

- Etiqueta que contiene el texto asociado al subtexto.

<!ELEMENT subimagen (#PCDATA)>

- Etiqueta que contiene la ruta de la imagen asociada al subpárrafo.

5.3 Formato de los documentos Base XML

Las etiquetas que se utilizan en los documentos XML, que almacenan las decoraciones generadas por la arquitectura, tienen el siguiente significado:

<documentoTextual>

- Etiqueta que contiene los datos representativos de un documento

<titulo>

- Etiqueta que contiene el título del documento

</titulo>

<capitulo>

- Etiqueta de inicio de capitulo del documento. Pueden existir más de un capitulo por documento

<titulocap>

- Etiqueta que representa el titulo de cada uno de los capítulos del documento. Solo puede haber un titulo por capítulo.

```
        </titulocap>
        <parrafo>
            - Etiqueta que representa el inicio de los sucesivos párrafos que
              poseen los documentos. Existen tantos párrafos como sean
              necesarios.
            <texto>
                - Etiqueta que representa el texto que puede poseer cada
                  párrafo. Solo existe una por párrafo, ya que un nuevo
                  texto pertenece a un nuevo párrafo.
            </texto>
            <imagen>
                - Etiqueta que representa la dirección donde se encuentra
                  la imagen asociada. Solo puede haber una imagen por
                  párrafo.
            </imagen>
            <subparrafo>
                - Etiqueta que representa los diversos subpárrafos que
                  pueden poseer los párrafos. Pueden existir tantos
                  subpárrafos como se desee.
                <subtitulo>
                    - Etiqueta que representa los diversos subpárrafos que
                      pueden poseer los párrafos. Pueden existir tantos
                      subpárrafos como se desee.
                </subtitulo>
                <subtexto>
                    - Etiqueta que representa el texto asociado a cada uno
                      de los subpárrafos. Puede haber tantos como se
                      desee.
                </subtexto>
                <subimagen>
                    - Etiqueta que representa una imagen que se
                      encuentre en el subpárrafo. Puede no existir.
                </subimagen>
            </subparrafo>
        </parrafo>
    </capitulo>
</documentoTextual>
```

5.4 Formato de los documentos Decorados XML

Las etiquetas que describe un documento base XML, tienen el siguiente significado, dependiendo del tipo de decoración del que se trate:

En el caso de un resumidor:

<decoracion tipo="0" nombre="ResumidorClassifier4j" >
- Etiqueta que contiene la decoración del resumidor con su identificador de tipo de decorador igual a 0, así como el nombre del resumidor concreto que generó dicho resumen. De este modo se puede añadir diversos resúmenes de distintos resumidores.

</decoracion>

En el caso de un extractor:

<decoracion tipo="1" nombre="ExtractorErial">
- Etiqueta que contiene la decoración del extractor con su identificador de tipo de decorador igual a 1, así como el nombre del extractor concreto que generó dicha extracción. De este modo se puede añadir diversas extracciones de distintos extractores.

</decoracion>

En el caso de un traductor:

<decoracion tipo="2" nombre="TraductorGoogle" idioma="en|fr">
- Etiqueta que contiene la decoración del traductor con su identificador de tipo de decorador igual a 2, así como el nombre del traductor concreto que generó dicha traducción. El idioma indica como primer parámetro el idioma del documento origen y el segundo parámetro indica el idioma al que se ha traducido. Estos parámetros son dependientes de la clase que implementa el traductor.

</decoracion>

En el caso de un clasificador:

<decoracion tipo="3" nombre="ClasificadorWekaSimple" >
- Etiqueta que contiene la decoración del clasificador

</decoracion>

5.5 Implementación de la arquitectura

Es necesario destacar ciertos aspectos que han implementado para de este modo facilitar el entendimiento del sistema.

Por un lado, fue necesario incluir un identificador de fichero, que aparece concretamente en el formato del fichero XML del documento base. Esto es así para de evitar que pudiesen incluir un mismo documento original al sistema pero con nombres diferentes. Esto se consiguió mediante el uso de la librería fast-MD5_2.6, disponible en la dirección http://www.twmacinta.com/myjava/fast_md5.php, que implementa un algoritmo de hash MD5 totalmente escrito en Java. MD5 es el acrónimo de "*Message-Digest Algorithm 5*, que es un algoritmo de reducción criptográfico de 128 bits ampliamente usado. De este modo, incluyendo un sencillo código en la implementación en el que solo se le pasa la ruta del fichero, para cada uno de ellos devolverá un hash diferente. En caso de que el usuario quisiera añadir documentos con diferentes nombres pero con contenido idéntico, el sistema solo almacenaría uno de ellos, ya que existiría un identificador igual asignado ya a un documento.

```
byte[] buf = new byte[65536];
MD5InputStream in = new MD5InputStream(new BufferedInputStream(new
                    FileInputStream(nombrefichero)));
while (in.read(buf) != -1);
String hash = MD5.asHex(in.hash());
in.close();
```

Otra aspecto que es necesario destacar es como se implementó el patrón decorador. A continuación se añaden fragmentos de código que ilustran dicha implementación.

```
public interface DocumentoXML{
   public String manejarDocumentoXML(DecoracionStructure dec,
   DocumentoXML doc);
   …
}
```

Define la interfaz de los objetos que tienen como responsabilidad añadirse dinámicamente.

```
public class DocumentoBase implements DocumentoXML
{
   public String manejarDocumentoXML(DecoracionStructure dec,
   DocumentoXML doc)
   {
```

```
    // crea si es documento base y si proviene de una decoración se añade
    recursivamente.
        ....
    }
    ...
}
```

Define un objeto al cual le pueden ser añadidas responsabilidades adicionales.

```
public class DecoradorDocumento implements DocumentoXML
{
    private DocumentoXML doc;
    public void setDocumentoXML( DocumentoXML d ){
        this.doc = d;
    }
    public DocumentoXML getDocumentoXML(){
        return this.doc;
    }
    public String manejarDocumentoXML(DecoracionStructure dec,
    DocumentoXML doc){
        return this.docXML.manejarDocumentoXML(dec,doc);
    }
...
}
```

Esta clase mantiene una referencia a un objeto DocumentoXML y define un interfaz que se conforme con el interfaz del DocumentoXML.

Otro de los temas que es necesario destacar es como se ha generado la visualización de los documentos. El código que se muestra corresponde a la generación del código HTML mediante XSLT, haciendo uso de una hoja de estilos como se ilustra en el siguiente fragmento de código.

```
public void VisualizacionDocBase (String fichero) throws MyException{
    File datafile = new File(fichero);
    if(!fdatafile.exists()){
        DocumentBuilderFactory factory
=DocumentBuilderFactory.newInstance();
        try {
            File stylesheet =new File (new URI ( "file:"+
            getClass().getResource("/").getPath()+"/arquitex/VisualizacionDocum
            entos/xml2html
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(datafile);
            TransformerFactory tFactory = TransformerFactory.newInstance();
            StreamSource stylesource = new StreamSource(stylesheet.getPath());
            Transformer transformer = tFactory.newTransformer(stylesource);
```

```
                DOMSource source = new DOMSource(document);
                StreamResult result = new StreamResult(datafile);
                transformer.transform(source, result);
        }
        catch (Exception tce) {
                throw new MyException (tce.getMessage() );
        }
}
```

El siguiente código representa la hoja de estilo del XSLT necesario para generar las páginas html:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0" >
   <xsl:output method="html"/>
   <xsl:template match="/">
        <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="/documentoTextual/titulo">
        <h1><font color="#FF6600" size="2" face="Verdana, Arial, Helvetica,
sans-serif">
                <xsl:apply-templates/>
        </font></h1>
   </xsl:template>
   <xsl:template match="documentoTextual/capitulo">
        <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="capitulo/titulocap">
        <h3><font color="#003366" size="1" face="Verdana, Arial, Helvetica,
sans-serif"><strong>
                <xsl:apply-templates/>
        </strong></font></h3>
   </xsl:template>
…
</xsl:stylesheet>
```

Por cuestiones de eficiencia, se consideró que las presentaciones de los documento se debían de crear en el momento en el que se crease el documento base (en caso de ser un documento base) o los documentos decorados (en el caso de ser documentos decorados). De este modo, el usuario en el momento de querer visualizar un documento evitará el retardo necesario para poder generar su presentación.

5.6 Pruebas de funcionamiento de la arquitectura

Durante el desarrollo del software, existen un gran número de posibilidades de error, por lo tanto, este proceso debe de ir acompañado de una actividad que garantice la calidad. Por este motivo se realizan las pruebas sobre el sistema. Gran parte del esfuerzo que se dedica al desarrollo, se invierte luego en esta fase, que debe de realizarse durante todas las etapas del desarrollo.

El motivo por el que se realizan estas pruebas, es diseñar aquellas que tengan mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo posible. Estas pruebas se van a seguir realizando mientras existan errores detectables, ya que no aseguran la ausencia de esos errores.

Según Pressman[7]:

- Los errores intentan encontrar errores.
- Un buen caso de pruebas es aquel que tiene una alta posibilidad de encontrar un error que no fue descubierto hasta el momento.
- Una prueba tiene éxito si se encuentra un nuevo error, a diferencia de lo que se pensaba hasta el momento, que basaban el éxito en el hecho de no encontrar errores.

5.6.1 Pruebas de Unidad

Se tiene que probar que la información que fluye entre cada módulo se hace de forma adecuada hacia y desde la unidad de programa que está siendo probada, para ello, se tiene que probar cada componente por separado.

5.6.1.1 Pruebas de Caja Blanca

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

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

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

5.6.1.2 Pruebas de Caja Negra

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

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

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

- Se implementó una clase cargadora de documentos, "*ConvertTXT2XML*", para probar el funcionamiento del sistema cuando se añadían documentos a la colección. De este modo se verificó que la información era extraída y el sistema era capaz de crear documentos base con formato XML bien formados.
- Se implementó una clase por cada una de las subclases de "*DecoradorDocumento*" para probar el funcionamiento del sistema cuando se solicitaba aplicar el decorador sobre la visión que indicaba el fichero de configuración. De este modo se verificó que era capaz de crear los documentos decorados con formato XML adecuado a los parámetros indicados para cada uno de los servicios sobre documentos individuales indicado en el fichero de configuración. Las clases concretas implementadas son "*TraductorGoogle*", "*ExtractorErial*", "*ClasificadorWekaSimple*","*ResumidorClassifier4j*".
- Se implementó una clase por cada una de las distintas subclases de "*Servicios*" para probar el funcionamiento del sistema cuando se solicita realizar algún tipo de búsqueda sobre los servicios implementados. Para la verificación se efectuaron distintas pruebas comprobando que los documentos devueltos se ajustaban a la consulta realizada. Además, con las clases implementadas, se definió una cadena de servicios para comprobar de este modo que el sistema era capaz de realizar una consulta tras otra para finalmente devolver el resultado correcto. Las clases implementadas se llaman "*IndexadorLucene*","*ClusterWekaSimple*"

Con ello se comprobó que todas las funciones definidas en la especificación de requerimientos habían sido implementadas y que cada una de ellas realizaba correctamente acciones que obtenían resultados deseados.

5.6.2 Pruebas de Integración

El objetivo principal de las pruebas de integración es detectar errores en la interacción entre las distintas clases que componen el sistema. En este sentido, durante la implementación del sistema se han ido validando las clases de Java utilizado para una vez completado el desarrollo, realizar una prueba del sistema global, obteniendo en ambos casos resultados satisfactorios.

5.6.3 Pruebas de Validación

Se han realizado múltiples pruebas para asegurar que los documentos XML existentes en el sistema sean válidos. La validación de estos documentos se centra en su estructura. Se ha comprobado que los documentos XML estén estructurados de acuerdo al DTD asociado. Este DTD se encuentra especificado en la página 180.

5.6.4 Pruebas de carga del sistema

En el momento de insertar una gran cantidad de documentos en el sistema, del orden de unos 2000 documentos, y de configurarlo de tal modo que se tenga que crear numerosas operaciones sobre los documentos, es decir que tenga que crear numerosas visiones y adjuntarlos a numerosos servicios sobre grupos de documentos, se observó que Java devolvía un error ya que no era capaz de soportar tanta carga en la pila.

Después de consultar la documentación de *Weka*, sus desarrolladores informan de la necesidad de aumentar el tamaño de la pila de la máquina virtual, ya que el hecho de devolver un mensaje "*OutOfMemory*"es bastante habitual debido a que *Weka* realiza la construcción de árboles recursivos con bastante profundidad como para provocar este tipo de situación, por lo que recomiendan aumentar el tamaño de la pila hasta 1024Mb para 1 Gb.

De todos modos aunque no se incluya en el fichero de configuración el algoritmo de clasificación o el cluster *Weka*, si se incorporan numerosos servicios tanto sobre documentos individuales y de grupos de documentos mientras se trabaja con numerosos documentos en la colección, podría suceder este tipo de error.

La solución adoptada fue incrementar la pila de java mediante: *java –mx1024M –jar weka.jar*

5.7 Pruebas de funcionamiento de Arquitex

5.7.1 Pruebas de Unidad

Las pruebas de unidad que se han realizado sobre el sistema Arquitex, son las que se van a detallar a continuación.

5.7.1.1 Pruebas de Caja Negra

Añadir Documentos:

TITULO:	SELECCIONAR DOCUMENTO CON EXTENSIÓN ERRÓNEA
Descripción:	Se selecciona un documento para subir, cuya extensión no es ni .html, ni .doc, ni .txt ni .pdf
Resultado:	Arquitex muestra un mensaje de error, indicando los tipos de documentos que puede añadir a la colección
Evaluación:	Correcto

TITULO:	NO SELECCIONAR NINGÚN DOCUMENTO
Descripción:	No se selecciona ningún documento, es decir el campo está vacío.
Resultado:	Arquitex muestra un mensaje de error, indicando que para añadir un documento a la colección es necesario indicar el fichero
Evaluación:	Correcto

TITULO:	CANCELAR ANTES DE AÑADIR EL DOCUMENTO
Descripción:	Se decide cancelar la operación, aun habiendo seleccionado el fichero.
Resultado:	Arquitex redirecciona a la página de inicio
Evaluación:	Correcto

Eliminar Documentos:

TITULO:	NO SELECCIONAR NINGÚN DOCUMENTO A ELIMINAR
Descripción:	No se selecciona ningún documento para ser eliminado
Resultado:	Arquitex muestra un mensaje de error, indicando que no se ha seleccionado ningún documento para poder realizar la eliminación
Evaluación:	Correcto

TITULO:	CANCELAR ANTES DE ELIMINAR EL DOCUMENTO
Descripción:	Se selecciona algún documento pero se decide cancelar la operación.
Resultado:	Arquitex redirecciona a la página de inicio.
Evaluación:	Correcto

Configurar Herramienta: ventana 1

TITULO:	INTRODUCIR ESPACIOS EN BLANCO EN LOS DIRECTORIOS
Descripción:	Se rellenan los campos de los directorios con alguno que otro espacio en blanco.
Resultado:	Arquitex elimina esos espacios en blanco, ya que no es un símbolo permitido para la dirección de ficheros.
Evaluación:	Correcto

Este mismo caso se puede aplicar también a los cargadores, a los servicios sobre documentos individuales, a los servicios de grupos de documentos y las cadenas de servicios. En estos casos, ya que se vana a tener que insertar estos parámetros en el fichero de configuración, el elemento separador será la ",".

TITULO:	NO RELLENAR ALGUNO DE LOS CAMPOS DE LOS CARGADORES
Descripción:	No se rellenan los campos de los nombres de los cargadores del sistema.

Resultado:	Arquitex muestra un mensaje de error indicando que es necesario incluir al menos uno de ellos.
Evaluación:	Correcto

TITULO:	NO RELLENAR ALGUNO DE LOS CAMPOS DE LOS SERVICIOS SOBRE DOCS INDIVIDUALES
Descripción:	No se rellenan los campos de los nombres de los servicios sobre documentos individuales.
Resultado:	Arquitex muestra un mensaje de error indicando que es necesario incluir al menos uno de ellos.
Evaluación:	Correcto

TITULO:	NO RELLENAR ALGUNO DE LOS CAMPOS DE LOS SERVICIOS DE GRUPOS DE DOCUMENTOS
Descripción:	No se rellenan los campos de los nombres de los servicios de grupos de documentos.
Resultado:	Arquitex muestra un mensaje de error indicando que es necesario incluir al menos uno de ellos.
Evaluación:	Correcto

TITULO:	CANCELAR LA OPERACIÖN DE LA CONFIGURACIÓN
Descripción:	Se cancela la operación en el medio de los cambios en esta primera etapa.
Resultado:	Arquitex redirecciona a la pagina de inicio.
Evaluación:	Correcto

TITULO:	SEGUIR LA CONFIGURACIÓN
Descripción:	Se pulsa la opción de siguiente para pasar al paso de 2 de la configuración.
Resultado:	Arquitex no registra los cambios realizados en el primer paso y muestra el paso 2.
Evaluación:	Correcto

Configurar Herramienta: ventana 2

TITULO:	INTRODUCIR CLASES INEXISTENTES EN LOS VISUALIZADORES
Descripción:	Se rellenan alguno de los campos de los servicios con un parámetro visualizador que no es una clase.
Resultado:	Arquitex muestra un mensaje de error indicando que la clase insertada no es válida.
Evaluación:	Correcto

Este mismo caso se puede aplicar también al parámetro clase de los servicios, por lo que no se repite.

TITULO:	DEJAR ALGÚN CAMPO SIN RELLENAR
Descripción:	No se rellenan todos los campos del formulario.
Resultado:	Arquitex muestra un mensaje de error indicando que es necesario rellenar todos los campos.
Evaluación:	Correcto

TITULO:	INSERTAR ESPACIOS EN BLANCO EN ALGÚN CAMPO
Descripción:	Se rellena alguno de los campos del formulario con espacios en blanco.
Resultado:	Arquitex automáticamente elimina esos espacio en blanco ya que no existen clase que los posean.
Evaluación:	Correcto

TITULO:	INSERTAR VALOR INCORRECTO EN EL CAMPO TEMPORAL DEL SERVICO DE GRUPOS DE DOCUMENTOS
Descripción:	Se rellena algún campo del parámetro temporal de un servicio de grupos de documentos con valores distintos a “verdadero“ o “falso”.
Resultado:	Arquitex muestra un error, indicando que solo que permite “verdadero” o “falso”.
Evaluación:	Correcto

TITULO:	INSERTAR VALOR DISTINTO DE ALGUN SERVICIO SOBRE DOCS INDIVIDUALES EN EL PARAMETRO VISTALISTA
Descripción:	Se rellena algún campo del parámetro vistaLista con un valor que no pertenezca a alguno de los servicios sobre documentos individuales que indicó.
Resultado:	Arquitex muestra un error, indicando que vistaLista no es correcto.
Evaluación:	Correcto

TITULO:	INSERTAR VALOR DISTINTO A UN ENTERO EN NUMPASOS
Descripción:	El valor insertado en el numero de pasos de la cadena no es un entero.
Resultado:	Arquitex muestra un error, indicando que solo se permite enteros.
Evaluación:	Correcto

Configurar Herramienta, ventana 3:

TITULO:	INSERTAR VALOR PARAMETROS PROPIOS DE SERVICIOS INCORRECTO
Descripción:	El valor insertado en los parámetros propios de los servicios es incorrecto.
Resultado:	Arquitex muestra un error, indicando el mensaje que implementó el usuario.
Evaluación:	Correcto

Configurar Herramienta, ventana 4:

TITULO:	INSERTAR NOMBER DE SERVICOS DE GRUPOS EN CADNE INCORRECTOS
Descripción:	El valor insertado en alguno de los pasos de la cadena de servicio es incorrecto, no es un servicio de grupos de

	documentos.
Resultado:	Arquitex muestra un error, indicando que uno de los pasos de la cadena de servicios es incorrecto.
Evaluación:	Correcto

TITULO:	INTRODUCIR CLASES INEXISTENTES EN LAS CLASES DE OTROS
Descripción:	Se rellenan alguno de los campos de "otros" con un parámetro que no es una clase.
Resultado:	Arquitex muestra un mensaje de error indicando que la clase insertada no es válida.
Evaluación:	Correcto

Este mismo caso se puede aplicar también al parámetro clase de los servicios, por lo que no se repite.

Búsqueda Avanzada:

TITULO:	NO INTRODUCIR NINGUNA CONSULTA EN UN INDEXADOR O UNA CADENA DE SERVICIOS
Descripción:	No se indica cual es la consulta que se quiere realizar sobre el servicio sobre el que se está tratando.
Resultado:	Arquitex muestra un mensaje de error indicando que es necesario realizar una consulta no vacía.
Evaluación:	Correcto

5.7.2 Pruebas de Integración

Para cada uno de los módulos probados en la prueba de unidad, se comprueba que se integra perfectamente con el resto de módulos formando el sistema final. Se prueban los enlaces entre as distintas páginas JSP. Se realizaron de la misma forma que a última parte de las pruebas de unidad, pero comprobando que se pasa correctamente de una página a otra.

6. Problemas, Conclusiones y Ampliaciones

6.1 Problemas Encontrados

Durante el desarrollo de este proyecto han surgido diversos problemas. A continuación se indican cuales han sido los más relevantes y como se han podido solucionar.

El primer problema encontrado ha sido la incompatibilidad entre las versiones del JDK 1.5 de Java y la versión 3.0 de Eclipse en Linux, que impedía la ejecución de Eclipse con esta configuración. Tras diversas pruebas, se solucionó el problema instalando una versión anterior de la maquina virtual de Java, en concreto el JDK 1.4.2.

Un problema similar de incompatibilidad de versiones impedía que se ejecutasen las nuevas clases implementadas en Java desde paginas JSP. En la versión de Tomcat utilizada, únicamente se permitía la ejecución de Java Beans y de clases estándar de Java, en cambio no era posible realizar la carga dinámica de las clases en tiempo de ejecución que se usan en la arquitectura. El error era consecuencia de la incompatibilidad de la versión JDK 1.4.2 de Java sobre la que se ejecutaba y la versión 4.1 el Tomcat. Tras varias pruebas, el problema se solucionó instalando una versión posterior del servidor Tomcat, concretamente la versión 5.0

Además de lo expuesto anteriormente, otra dificultad añadida han sido las interpretaciones que se pueden dar a la hora de implementar los distintos patrones de diseño utilizados en este trabajo. Existen diversas formas de programar un mismo patrón y se debe buscar la más idónea para el caso que nos ocupa. Por lo que ha sido necesario evaluar y realizar pruebas de cual era la implementación que más se aproximaba a las necesidades que se requerían. Por ejemplo, en un primer intento el patrón decorador se implementó con la clase DocumentoXML como una clase abstracta, dada que esta aproximación presentaba problemas, se optó por usar una interfaz.

Otro de los problemas detectados se registro en las pruebas de carga del sistema, donde una carga muy elevada de documentos un provocó el consumo excesivo de recursos en el sistema, en concreto se alcanzaba la saturación de la pila de ejecución de la máquina virtual Java. Las posibles soluciones que se pueden adoptar son:

- Marcar los objetos que ya no se estén usando forzándolos a ser eliminados por el recolector de basura
- Ejecutar el recolector de basura de java cuando la pila alcance valores de ocupación alto
- Reducir el almacenamiento de elementos en memoria. En la implementación actual se ha optado por trabajar mayoritariamente sobre

datos almacenados en memoria para mejorar los tiempos de respuesta del sistema pero, a cambio, se produce un excesivo consumo de recursos. Lo optimo sería llegar a un equilibrio entre consumo de recursos y rendimiento del sistema. Una solución eficiente pasa por estudiar cuidadosamente el uso del almacenamiento en memoria para eliminar la carga de determinados archivos en memoria, trabajando con éstos directamente desde el disco. La contrapartida de esta aproximación es que producirá una excesiva ralentización de todos los procesos.

Además de todos los problemas anteriores, cabe destacar que el uso del navegador Firefox para la presentación de los documentos HTML en el cliente produjo también ciertos problemas con las funciones de Javascript, ya que las distintas interpretaciones que hacen los navegadores del código javascript hace que sea necesario en algunos casos implementar soluciones a medida del mismo problema.

Finalmente, el incluir un sistema de precarga en html sin tener que desarrollar un applet que controle los procesos que se están ejecutando no fue tarea fácil. Este problema surge cuando el servidor debe enviar una página cuyo preprocesamiento es muy elevado, algo relativamente frecuente en determinados servicios. El usuario se queda esperando por un contenido que a veces tarda en llegar sin saber si realmente se está ejecutando o no. La solución adoptada finalmente radica en un pequeño engaño al usuario: se ejecuta previamente una página de espera y a continuación se invoca a la página que realiza el proceso en el servidor. Cuando el servidor acabe de procesar la página, la enviará al cliente sustituyendo la página de precarga o espera.

6.2 Conclusiones

La principal conclusión a la que se ha llegado después de desarrollar este sistema, es que muchas de las herramientas que se encuentran en estos momentos disponibles están dirigidas a realizar unas tareas determinadas, específicas de cada una de ellas, como es el caso de Lucene, cuya funcionalidad básica es realizar la indexación y de Weka, como clasificador y cluster. Me he encontrado con la dificultad de enfrentarme a un proyecto donde se deben de integrar muy diversas tecnologías que realicen trabajos independientes y llevarlas a una arquitectura común. Resultó complicado pronosticar los resultados que finalmente se obtendrían, ya que se desconocían los posibles efectos colaterales que podían existir en la integración, además de que no existir actualmente ninguna arquitectura que pudiera servir como referencia y que garantizase el éxito del desarrollo.

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

Es necesario destacar la dificultad para ir estableciendo límites durante el desarrollo del proyecto debido a su envergadura ya que a medida que se avanzaba, se iban identificando nuevas mejoras que se podían incorporar.

Uno de los retos más importantes que se han encontrado a la hora de realizar este proyecto, ha sido el gran número de tecnologías y herramientas empleadas: HTML, JSP, XML, XSL, Java, JavaScript, herramientas y tecnologías estándar en el ámbito del procesamiento de lenguaje natural (PLN) y en el de la recuperación de información (RI). Fue necesario dedicar cierto tiempo en el estudio y el manejo de estas tecnologías, sobre todo en lo que respecta al uso de las diversas herramientas como Weka, Lucene, Classifier4j y Erial.

El incorporar tantas herramientas en la arquitectura provocó una serie de complicaciones durante el desarrollo tanto de la arquitectura, como de la aplicación Arquitex. Resultó especialmente difícil encontrar la fuente de algunos errores a causa de la disparidad de tecnologías empleadas. Entre ellos destacan las dificultades procedentes de gestión de clases cargadas dinámicamente y en tiempo de ejecución. Todo ello ayudó un a adquirir mayor conocimiento de las tecnologías empleadas.

Finalmente, hay que resaltar la necesidad de realizar pruebas de carga exhaustivas al sistema en fases tempranas del desarrollo. Se ha comprobado como tratar de aumentar la eficiencia o depurar determinado tipo de errores en fase tardías del desarrollo tienen un coste muy elevado.

6.3 Ampliaciones

Con este proyecto se ha pretendido desarrollar una arquitectura lo suficientemente flexible como para ser aplicada a cualquier tipo de ámbitos. El trabajo desarrollado pese a tratarse de una primera aproximación, es capaz de dar soporte a un buen número de aplicaciones y ofrece una muy buena base para continuar el desarrollo e incorporar nuevas funcionalidades.

En lo que respecta a posibles ampliaciones de este proyecto en un futuro, estas se pueden organizar del siguiente modo:

1. Mejora de los módulos más importantes del sistema, en el sentido de:

    - Permitir trabajar con más de una colección, de este modo se podrían realizar búsquedas sobre varias colecciones a la vez.
    - Realizar búsquedas sobre varios servicios de grupos de documentos, sin hacer uso del encadenamiento de servicio.
    - Incorporar documentos a la colección que no sean solo txt, doc, html o pdf sino abarcar una mayor cantidad de formatos según las necesidades del usuario.
    - Implementar métodos de carga de documentos más sofisticados. Que podrían soportar, por ejemplo, la descarga periódica de documentos de origen desde fuentes remotas, como pueden ser los textos de publicaciones

oficiales (BOE,DOG) el otro tipo de publicaciones electrónicas, como periódicos on-line.

- Eliminar la restricción que se estableció en el encadenamiento de servicios, es decir, permitir que el clustering se pueda incorporar en cualquiera de los pasos y que este no tenga porque ser un paso final. De este modo se dotaría al sistema de mayor flexibilidad.
- Incorporar a la arquitectura un mayor número de presentaciones por defecto como podría ser la visualización de los documentos y de los resultados mediante el uso de *frames*.
- Implementar la paginación en la visualización de los resultados de los servicios de grupos de documentos.
- Permitir al usuario escoger el formato de los documentos XML, para de este modo procesar documentos escritos en idiomas que no posean caracteres ISO como podría ser el caso de los documentos escritos en chino.

2. Aplicación del sistema a otros entornos, incluyendo el tratamiento de otros tipos de información como pueden ser las imágenes, la multimedia etc.

3. Incorporar un sistema de registro de incidencias que permita almacenar en ficheros de logs todos los sucesos que ocurran como consecuencia de actuaciones incorrectas sobre el sistema o situaciones anómalas.

7. Apéndice Técnico

7.1 Apéndice A: Indexación de la información – Lucene

7.1.1 Introducción

Lucene es una novedosa herramienta que permite tanto la indexación como la búsqueda de documentos. A pesar de solo trabajar con texto, posee otros añadidos que permiten indexar documentos de Word, ficheros PDF, XML o páginas HTML. Creada e implementada completamente en Java, se trata de una API flexible, muy potente y fácil de usar, a través de la cual se pueden añadir, con pocos esfuerzos de programación, capacidades de indexación y búsqueda a cualquier sistema que se esté desarrollando.

Originalmente escrita por Doug Cutting en septiembre de 2001pasó a formar parte de la familia de código abierto de la fundación Jakarta. Además de Lucene, existen otras herramientas que permiten realizar indexación y búsquedas de documentos pero que han sido optimizadas para usos concretos, lo que implica adaptarlas al proyecto que aquí se trata. La idea que engloba Lucene es completamente diferente ya que su principal ventaja es su flexibilidad, que permite su utilización en cualquier sistema que lleve a cabo procesos de indexación, proporcionando un "framework" básico.

7.1.2 Características

A continuación se detallan algunas de las características que hacen de Lucene una herramienta flexible y adaptable:

- **Indexación incremental vs. indexación por lotes:**

  La indexación por lotes se refiere a aquellos procesos de indexación, en los cuales, una vez creado el índice para un conjunto de documentos, si se quieren añadir unos documentos nuevos es necesario reindexar todos los documentos de nuevo. En cambio la indexación incremental permite añadir nuevos documentos a un índice ya creado con anterioridad de forma fácil. Lucene permite ambos tipos de indexación.

- **Origen de datos:**

  Muchas herramientas de indexación solo permiten indexar ficheros o páginas Web. Lucene además de permitir indexar esos ficheros y páginas Web también permite indexar el contenido procedente de una base de datos.

- **Contenido etiquetado:**

Mientras algunas herramientas solo permiten indexar documentos tratándolos como simples flujos de palabras, Lucene permite dividir el contenido de los documentos en base a campos de información y poder así hacer consultas con mayor contenido semántico. Por ejemplo se podría dividir los documentos en dos campos, titulo y contenido, por lo que se podría realizar búsquedas sobre aquellos que contengan los términos de la búsqueda en el campo titulo.

- **Técnica de indexación:**

Existen palabras que no añaden significado al índice como pueden ser a, el, la, … etc. Por ser palabras poco representativas del documento. Al eliminarlas, el índice se reduce considerablemente así como el tiempo que tarda en realizarla. Estas palabras están contenidas en lo que se denomina lista de parada y es la técnica empleada por Lucene.

- **Elección del idioma:**

Tal y como se indicó en el apartado anterior, Lucene emplea las denominadas listas de parada que son proporcionadas por el desarrollador. Esto permite seleccionar el idioma que se va a utilizar.

- **Concurrencia:**

Lucene permite que varios usuarios busquen en el índice de forma simultánea así como que un usuario modifique el índice mientras otro lo consulta.

7.1.3 Como obtenerlo

Como la mayoría de los proyectos de Jakarta, Lucene se distribuye en formato de ficheros binarios precompilados o de código fuente. Ambas distribuciones pueden ser descargadas de la página oficial de Jakarta, http://lucene.apache.org/java/docs/, así como también una demo que permite ver el funcionamiento de Lucene.

7.1.4 Funcionalidad básica

A continuación, y puesto que la indexación y la búsqueda son dos operaciones muy generales, que abarcan múltiples aspectos, se tratan en detalle cada uno de ellos:

7.1.4.1 Indexación de documentos

Creación de un *Document* a partir de documentos concretos:

La creación de un *Document* a partir de documentos concretos permite un índice constituye el punto de partida. Una vez creado se irán añadiendo los nuevos documentos susceptibles de ser indexados.

```
private Document crearDocumento(String palabra, String direccionXML, String nbXML){
```

```
    Document document=new Document();
    document.add(Field.Text("titulo",palabra));
    document.add(Field.UnIndexed("direccion",direccionXML));
    document.add(Field.UnIndexed("nombre",nbXML));
    return document;
}
```

Lo primero que se hace es crear un nuevo objeto *Document.* Lo siguiente a hacer es añadir las diferentes secciones del documento concreto al *Document*. Los nombres que se dan en cada sección son completamente arbitrarios y funcionan como las llaves de un *HashMap*. Los nombres usados deben de ser *String*. El método *add* de *Document* tomará un objeto de tipo *Field*. Existen 4 métodos proporcionados por los objetos *Field* de un *Document*.

- ***Field.Keyword***: el dato es almacenado e indexado pero no es tokenizado. Es lo más utilizado para datos que deberían de ser almacenados sin cambiarse, como puede ser una fecha.
- ***Field.Text***: El dato es almacenado, indexado y tokenizado. *Field.Text* no se debe de usar para grandes cantidades de datos como podría ser el propio documento porque provocaría que el índice creciese en grandes proporciones ya que contendría una copia entera del documento y una versión tokenizada.
- ***Field.UnStored***: El dato no se almacena pero se indexa y se tokeniza. Grandes cantidades de información como pueden ser textos de los documentos se deben de situar en el índice no almacenado.
- ***Field.UnIndexed***: El dato es almacenado pero no indexado o tokenizado. Se usa con datos que se quiere que sean devueltos como resultado de una búsqueda pero que no se quiere buscar por ese campo.

Creación de un índice:

La creación de un índice constituye el punto de partida. Una vez creado se irán añadiendo los nuevos documentos susceptibles de ser indexados.

```
private void crearIndice(String indexDirectory) throws Exception {
    IndexWriter writer=new IndexWriter(indexDirectory, null, true);
    writer.close();
}
```

La clase IndexWriter se usa tanto para la creación como para el mantenimiento de un índice. Cuando se crea un objeto de este tipo, al constructor se le pasan tres parámetros. Como en este caso estamos creando el índice, solo son relevantes el primer y tercer parámetro: el primero representa el path donde se almacenará el índice una vez creado; el tercero indica que lo que estamos haciendo es justamente crear un índice y no abriéndolo para su mantenimiento.

El método *close()* se llama para liberar todos los recursos asociados a la creación del índice.

Añadir un documento a un índice

Una vez creado el índice, está listo para añadirle documentos.

```
String indexDirectory = "lucene-index";
public void addDocumentosIndice(Document doc) throws Exception{
    Analyzer analyzer = new StandardAnalyzer();
    IndexWriter writer = new IndexWriter(indexDirectory, analyzer, false);
    writer.addDocument(doc);
    writer.optimize();
    writer.close();
}
```

Primero se crea un *StandardAnalyzer* y luego el *IndexWriter* usando el analizador. En el constructor se debe de especificar el directorio donde el índice reside. El booleano al final del constructor indica que el *IndexWriter* está añadiendo documentos a un índice existente, por lo que se especifica false. Posteriormente se debe de añadir el objeto *Document* al índice. Finalmente, si se añaden múltiples objetos *Document*, se debería siempre optimizar y luego cerrar el índice.

El objeto *Analyzer* se crea para analizar un texto y en base a un determinado criterio, obtener la representación interna de dicho texto en el índice. Los analizadores preprocesan el texto de entrada convirtiendo éste en una secuencia de tokens y se utilizan para añadir un documento a un índice como en los procesos de búsqueda, puesto que el texto o criterio de búsqueda debe de ser procesado de la misma manera que el contenido de los campos del documento cuando éste sea añadido al índice.

Uno de los criterios más utilizados es el de la lista de parada, que consiste en eliminar del texto una serie de palabras que no resulten útiles para la obtención de términos tanto de indexación como de búsqueda. En este caso lo que se le pasa al constructor de *IndexWriter* es un objeto de *StopAnalyzer*, subclase de *Analyzer*.

Cualquier aplicación que use Lucene debe proporcionar los datos que van a ser indexados bien como *String* o bien como *InputStream*. Por este motivo permite indexar datos no solo de ficheros sino de cualquier fuente. En caso de que los documentos se encuentren almacenados en ficheros, se usa *FileInputStream* para recuperarlos. En caso de estar en una BD se usa *InputStream*.

7.1.4.2 Búsqueda de documentos

Una vez añadidos los documentos al índice, la búsqueda de documentos constituye la funcionalidad principal de Lucene.

```
private void BuscarDocumentos(String consulta,String indexDirectory) throws
Exception {
    IndexSearcher is = new IndexSearcher(indexDirectory);
    Analyzer analyzer = new StandardAnalyzer();
```

```
    QueryParser parser=new QueryParser("titulo",analyzer);
    Query query=parser.parse(consulta);
    Hits hits=is.serach(query);
    for(int i=0;i<hits.length();i++){
        Document doc=hits.doc(i);
        System.out.println(hits.doc(i).get("nombre")+"; Score: "+hits.score(i));
    }
    is.close();
}
```

Aunque hay gran cantidad de clases envueltas aquí, la búsqueda no es muy complicada. El primer paso es crear un objeto *IndexSearcher* apuntando a al directorio donde se encuentra el índice. Luego es necesario crear un objeto *Analyzer* que será pasado al constructor de *QueryParser* con el nombre del campo por defecto usado en la búsqueda. Este será el campo utilizado si el usuario no especifica el campo en su consulta. A continuación se parseará la consulta devolviendo un objeto de tipo *Query*. A partir de aquí se puede realizar la petición de búsqueda sobre el objeto *IndexSearcher*. Esto devolverá un objeto de tipo *Hits* que será una colección de documentos con el criterio de la búsqueda.

7.1.4.3 Formato de las queries

La clase *Query* es al igual que *Searcher* una clase abstracta, siendo *QueryParser* su subclase más importante. El método de esta clase que proporciona mayor funcionalidad es *parse()*. Este método partiendo de una cadena de entrada obtiene como parámetro de salida una Quero. Las queries están formadas por una serie de cláusulas de la forma:

- Toda cláusula debe de ir precedida de: un símbolo (+) o un símbolo (-) indicando si la cláusula es requerida o es rechazada; un termino seguido de una coma, indicando el campo donde se va a realizar la búsqueda, lo cual permite la construcción de Querys que implementen búsquedas en varios campos de un documento indexado.
- Además de un término, indicando todos los documentos que contienen dicho término o una Query anidada encerrada entre paréntesis.

En formato BNF, la gramática que resume esto sería:

```
Query ::= (Clause) *
Clause ::= ["+","-"] [<TERM> ":"] (<TERM> | "(" Query ")")
```

Lucene soporta un gran conjunto de posibilidades en las búsquedas incluyendo AND, OR y NOT, búsquedas por lógica difusa, búsquedas por proximidad y búsquedas por rango.

Ejemplo: Encontrar los documentos titulados "Desarrollo actual" y que posean la frase "teorías actuales" y que no contenga USA.

titulo: "Desarrollo actual" "teorías actuales" NOT USA

7.2 Apéndice B: Resumidor – Classifier4J

7.2.1 Introducción

Classifier4J es una librería de clasificación de textos implementado en Java que incluye un resumidor de textos y un clasificador Bayesiano. Este API es muy sencillo de usar y se puede incorporar tanto el resumidor como el clasificador Bayesiano en cualquier aplicación con muy pocas líneas de código.

7.2.2 Como obtenerlo

Como gran parte de los proyectos desarrollados de libre distribución, Classifier4j se distribuye en formato de ficheros binarios precompilados o de código fuente. Ambas distribuciones pueden ser descargadas de la página, http://classifier4j.sourceforge.net, así como también se tiene acceso a un pequeño manual sobre como usarlo de un modo muy sencillo.

7.2.3 Funcionalidad básica

Aunque en el proyecto que nos ocupa solo se ha implementado el resumidor, también se incluye en este apéndice un acercamiento de los métodos de clasificación que implementa esta librería.

7.2.3.1 Uso Básico

El modelo de uso básico del Classifier4J es el siguiente:

- Crear una instancia de la implementación de la interfaz *IClassifier.*
- Llamar o a *isMatch(String)* (para ocurrencia booleanas) o *classify(String)* .

El ejemplo más sencillo es el siguiente:

```
String palabra=”java”;
SimpleClassifier classifier=new SimpleClassifier();
Classifier.setSearchWord(palabra);
String frase=”This is a sentence about java”;
System.out.println(“The string “+sentence+” contains the word java”+classifier.isMatch(sentence));
```

La clase *SimpleClassifier* es una implementación de *IClassifier* la cual mira en el string pasado si existe la palabra que fue asignada mediante *setSearchWord(String)*.

7.2.3.2 Uso de BayesianClassifier

El *BayesianClassifier* es una implementación de la interfaz *IClassifier* que usa el teorema de Bayes sobre los textos de entrada.

```
IWordsDataSource wds= new SimpleWordsDataSource();
IClassifier classifier=new BayesianClassifier(wds);
System.out.println("Matches= "+classifier.classify("Esto es una frase");
```

Para algunas aplicaciones el *JDBCWordsDataSource* es más útil que el *SimpleWordsDataSource*. Se puede usar de un modo sencillo del modo que sigue:

```
DriverManagerJDBCConnectionManager cm=new
DriverManagerJDBCConnectionManager();
JDBCWordsDataSource wds=new JDBCWordsDataSource(cm);
IClassifier classifier =new BayesianClassifier(wds);
```

Sin embargo el entrenamiento del *JDBCWordsDataSource* es bastante pésimo. Si se desea mejorar el entrenamiento, se recomienda el uso de *JDBMWordsDataSource* (se encuentra en el Classifier4j-Optional download).

El clasificador bayesiano puede ser entrenado usando los métodos *teachMatch* y *teachNonMatch*. Para que el algoritmo funcione, es necesario entrenarlo con ambos (con los aciertos y los no-aciertos).

7.2.3.3 Uso de VectorClassifier

El *VectorClassifier* es una implementación de la interfaz *IClassifier* que usa el algoritmo de búsqueda de espacios vectoriales. Este algoritmo es bastante rápido (comparado con el algoritmo de Bayes) y no requiere del entrenamiento de los no-aciertos. También tiene la ventaja de que sus ratios de acierto (que son devueltos por el método de clasificación) están mejor distribuidos a pesar de que el clasificador Bayesiano tiende a devolver 0.99 o 0.01. Esta característica lo hace ideal para tareas de categorización.

Ejemplo sencillo:

```
TermVectorStorage storage=new HashMapTermVectorStorage();
VectorClassifier vc=new VectorClassifier(storage);
vc.teachMatch("categoría","hola, esto es una frase bastante larga bla bla bla");
double result=vc.classify("category","hola bla");
System.out.println(result);
```

Actualmente la desventaja es que una vez entrenado, es imposible añadir más entrenamiento a una categoría.

7.2.3.4 Uso de ISummariser

Usar el *ISummariser* es muy sencillo. Es necesario indicarle el texto que queremos resumir, y decidir en cuantas líneas queremos que se devuelva el resumen.

```
private String resumirDocumento(String input){
    Summariser summariser=new summariser();
    int lineas=2;
    String result=summariser.summarise(input,lineas);
    return result;
}
```

7.3 Apéndice C: Clasificación de la información – Weka

7.3.1 Introducción

WEKA (Waikato Environment for Knowledge Analysis) fue desarrollado en la universidad de Waikato en Nueva Zelanda. Se trata de un programa o entorno para el análisis de conocimientos. Está escrito en java por lo que se convierte en un sistema multiplataforma. Implementa numerosos algoritmos de aprendizaje y múltiples herramientas para transformar las bases de datos y realizar un exhaustivo análisis, como son tareas de clasificación, regresión, clustering, asociación y visualización. Weka está diseñado como una herramienta orientada a la extensibilidad por lo que añadir nuevas funcionalidades es una tarea sencilla.

La licencia de Weka es GPL, lo que significa que este programa es de libre distribución y difusión. Además, ya que Weka está programado en Java, es independiente de la arquitectura, ya que funciona en cualquier plataforma sobre la que haya una máquina virtual Java disponible.

WEKA se puede utilizar de 3 formas distintas:

- **Desde la línea de comandos:**

  Cada uno de los algoritmos incluidos en WEKA se puede invocar desde la línea de comandos de MS-DOS como programas individuales. Los resultados se muestran únicamente en modo texto.

- **Desde uno de los interfaces de usuario:**

  WEKA dispone de 4 interfaces de usuario distintos, que se pueden elegir después de lanzar la aplicación completa. Los interfaces son:
  - Simple CLI (command line interface): interfaz en modo texto.
  - Explorer: interfaz gráfico básico.

- Experimenter: interfaz gráfico con posibilidad de comparar el funcionamiento de diversos algoritmos de aprendizaje.
- KnowledgeFlow: interfaz gráfico que permite interconectar distintos algoritmos de aprendizaje en cascada, creando una red.

Para comprobar el aspecto de cada interfaz, se lanzará la aplicación. En el directorio donde se encuentre instalado WEKA, se deberá ejecutar el programa java weka.jar.

- **Creando un programa Java:**

La tercera forma en la que se puede utilizar el programa WEKA es mediante la creación de un programa Java que llame a las funciones que se desee. El código fuente de WEKA está disponible, con lo que se puede utilizar para crear un programa propio.

7.3.2 Características

A continuación se detallan algunas de las características:

- **Algoritmos de aprendizaje:**

De los muchos algoritmos de aprendizaje que WEKA implementa, cabe destacar los algoritmos cuya clasificación de datos está basada en árboles de decisión. En particular:

- según el análisis de los datos sea nominal:
  - **OneR**: Algoritmo de clasificación que genera un árbol de decisión de un único nivel. Es capaz de inferir reglas de clasificación a partir de un conjunto de instancias. Además, crea una regla para cada atributo en los datos de entrenamiento, luego escoge la regla con la tasa de error (es el número de instancias de los datos de entrenamiento en los que la clase del valor de un atributo no concuerda con la asociación que la regla le da al valor de ese atributo) más pequeño como su "one rule". Para crear una regla para cada atributo debe determinarse la clase más frecuente para cada valor del atributo.
  - **J48**: Se trata de una implementación propia de WEKA para el algoritmo C4.5, un algoritmo basado en clasificación por árbol de decisión. El algoritmo ofrece la posibilidad de poder parar antes de alcanzar las hojas en cada subárbol: PODA. Para ello se pueden usar dos parámetros para parar el algoritmo:
    - minNumObj (m): para, si el nº de elementos de un subconjunto es menor que m.
    - confidenceFactor: para, si la tasa de desclasificados en el subconjunto es menor que este umbral.

    Para seleccionar los nodos se basa en un sistema de penalización que consiste en añadir un término denominado “Split information” que desanima la selección de atributos con muchos valores distribuidos uniformemente.

- según el análisis de los datos sea numérico:
  - **Decisión Stump:** _ Consiste en la creación de un árbol binario de profundidad la unidad. Toda instancia inclasificable quedará colgada de una nueva rama que se une al nodo raíz. _ Parece obvio predecir que los errores que se cometerán a la hora de clasificar los datos serán elevados.
    - M5'

- **Conocimientos previos:**

La validación cruzada (cross- validation) consiste en el empleo de k subconjuntos del conjunto de datos, k/2 se emplean para entrenamiento y k/2 para la validación del esquema de aprendizaje. Cuando la validación se realiza con subconjuntos mezclados aleatoriamente se denomina validación cruzada estratificada. De este modo se consigue una clase determinada aparezca con la misma probabilidad en todos los subconjuntos de validación. _ Sólo es válido para conjuntos de datos nominales. Dentro de las distintas medidas que se ofrecen se prestará mayor atención al coeficiente de correlación que mide la correlación estadística entre los datos predecidos y los datos reales (proceso de validación del esquema).

7.3.3 Como obtenerlo

El programa WEKA se puede descargar desde:
http://www.cs.waikato.ac.nz/ml/weka

7.3.4 Funcionalidad básica

Existen más funcionalidades de la herramienta, pero las que aquí se describen son las que se usaron para el desarrollo de este proyecto.

7.3.4.1 Uso de Clasificador

Cualquier algoritmo en Weka deriva de la clase abstracta Classifier class. Asombrosamente poco es necesario para realizar un clasificador básico: una rutina que genera un modelo del clasificador de un dataset de entrenamiento y otra rutina que evalúe el modelo generado en un dataset no visto de la prueba, o genera una distribución de la probabilidad para todas las clases.

Para la realización de la clasificación de los documentos, es necesario como primer paso realizar un fichero que constará de lista de las palabras clave de las que esté formado los documentos calculando el peso que tienen asociado.

$$Peso_i = \log(1/n^o \text{ docs de entrenamiento donde aparece palabra clave } i)$$

Para cada documento que se procesa, se calcula la frecuencia de cada uno de los términos de la selección de palabras clave,

$$tf_{ij} = (\text{n de ocurrencias palabra clave i en el doc j} / \text{n palabras en doc j})$$

permitiendo de este modo generar para cada uno de los documentos, el vector asociado calculado por el peso $_i$ *tf_{ij}, siendo esto una aproximación de tf-idf(frecuencia inversa del documento)[1].

Hay que destacar que para la clasificación de un documento, es necesario entrenar Weka mediante un fichero ARFF. Para cada documento de la colección se extrajeron las palabras clave y a partir de los vectores extraídos y de las clases asignadas por el corpus reuters, disponible en la dirección http://daviddlewis.com/resources/testcollections/reuters21578/ , se construyó el fichero de entrenamiento ARFF necesario por Weka para realizar la tarea de clasificación.

Estos dos son por tanto los parámetros necesarios para el uso del clasificador Weka que se incorporó en la aplicación además de indicar cual va a ser la clase que va a realizar la tarea.

7.3.4.2 Uso de Cluster

Para el uso del cluster en la aplicación fue necesario el uso de un fichero que contuviese la lista de palabras clave, como se explicó en el apartado anterior y el uso de la clase necesaria para su realización. Además, también posee las opciones necesarias para su realización.

7.4 Apéndice D: Extractor de palabras clave – Erial

7.4.1 Introducción

Erial es un entorno para recuperación de información basado en técnicas de PLN y desarrollado en el marco del proyecto ERIAL, Extracción y Recuperación de Información mediante Análisis Lingüístico[11], un proyecto de investigación que aspira a mejorar los resultados de las aplicaciones de RI en gallego y español mediante el uso de técnicas de PLN. Ha sido desarrollado por un grupo de investigación de la Universidad de A Coruña que aportaba el conocimiento computacional, las Universidades de Santiago de Compostela y de Vigo y el centro Ramón Piñeiro para la investigación en Humanidades que aportaban el conocimiento lingüístico.

7.4.2 Características

A continuación se describe de manera resumida las principales características de los módulos que componen el sistema Erial.

- **Preprocesador:**

El primer paso en el procesamiento tanto de documentos como de consultas consiste en preprocesar el texto para, por una parte, segmentarlo adecuadamente, y por otra, para realizar diversas transformaciones en el mismo y así facilitar el trabajo de las fases posteriores. El concepto lingüístico de palabra no siempre coincide con su realización grafica. Por ello, Erial posee un segmentador avanzado que realiza tareas de preetiquetación de los textos sobre secuencias de caracteres no ambiguas tales como números o fechas, al tiempo que trata fenómenos como abreviaturas, acrónimos, contracciones, pronombres enclíticos, identificación de expresiones, etc.

- Fases del preprocesador

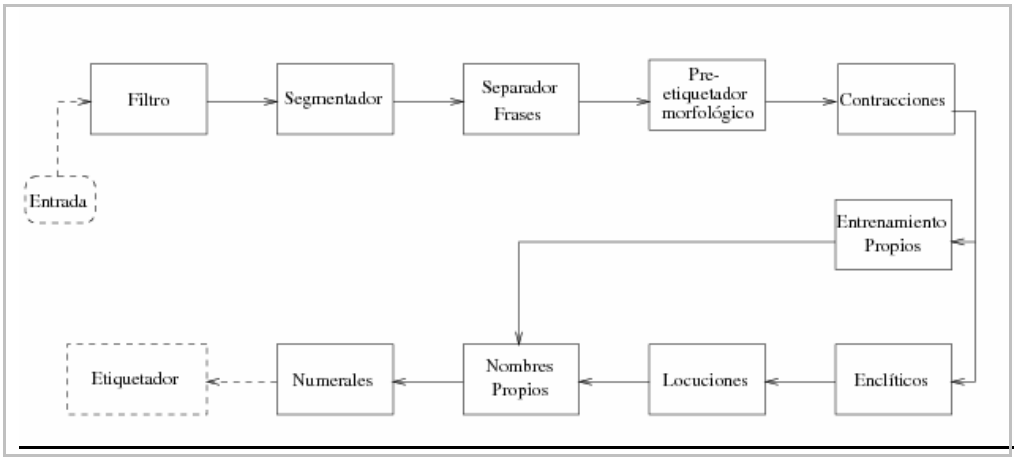


Figura 194: Fases del preprocesador Erial

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

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

El *modulo de separador de frase* es el que delimitar las frases, tarea que aunque es aparentemente sencilla, presenta numerosas dificultades. La regla general consiste en separar una frase cuando hay un punto seguido de

mayúscula, pero se deben tener en cuenta las abreviaturas para no segmentar después de un punto de alguna abreviatura especial.

El modulo preetiquetador morfológico es el que etiquetar elementos cuya etiqueta se puede deducir a partir de la morfología de la palabra y no existe una manera mas fiable de hacerlo.

El modulo de contracciones se encarga de desdoblar una contracción en sus diferentes tokens, etiquetando además cada uno de ellos. Para ello utiliza información externa que especifica cromo se descomponen las contracciones, de manera que sólo es necesario cambiar esta información para poner en funcionamiento este módulo sobre otro idioma.

El modulo de pronombre enclíticos se encarga de analizar los pronombres enclíticos que aparecen en las formas verbales.

El *modulo de locuciones* se encarga de unir los tokens que componen una locución y de etiquetarlos como una unidad conjunta.

El *modulo de entrenamiento de propios* constituye una primera fase para el tratamiento de los nombres propios. Su salida será utilizada por el siguiente modulo para identificar y etiquetar correctamente estos elementos del texto.

El *modulo de nombres propios*, a partir del diccionario que se ha generado en el modulo anterior, y a partir de los nombres propios que se puede tener en un diccionario externo, etiqueta los nombres propios, tanto simples como compuestos.

El *modulo de los numerales* se encarga de unir numerales para generar un numeral compuesto.

- **Etiquetador:**

Tras segmentar y preprocesar el texto, el siguiente paso consiste en etiquetarlo y lematizarlo. Con ello conseguimos obtener los rasgos morfosintácticos mas relevantes de cada palabra, información que será utilizada en las siguientes etapas, al tiempo que logramos eliminar la variación flexiva, normalizando las distintas realizaciones de verbos, sustantivos y adjetivos, a una sola forma, su lema La etiquetación de las partes del discurso se realiza mediante un modelo oculto de Markov (HMM) de segundo orden[1].

- Características de los modelos

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

- Manejo de palabras desconocidas.

Dada una palabra desconocida, se establece su conjunto de etiquetas candidatas, así como sus probabilidades asociadas, mediante el estudio de

los sufijos de la palabra. La distribución de probabilidad de un determinado sufijo se genera a partir de todas las palabras del corpus de entrenamiento que comparten el mismo sufijo, considerando una longitud máxima predefinida.

- Integración de diccionarios externos:

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

- Manejo de segmentaciones ambiguas.

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

- **Analizador sintáctico superficial:**

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

En general. uno de los métodos más utilizados es la denominada simplificación del texto: se comienza con el *stemming* de las palabras individuales junto con la eliminación de las *stopwords*, para posteriormente extraer y normalizan los términos relacionados empleando técnicas estadísticas. Existe, pues, una clara falta de base lingüística en dichas operaciones, lo que redunda frecuentemente en simplificaciones erróneas. Existen otros métodos con base lingüística que realizan un análisis sintáctico del texto, el cual devuelve a su salida un conjunto de árboles que denotan relaciones de dependencia entre las palabras involucradas. De este modo, estructuras con relaciones de dependencia similares pueden ser normalizadas a una misma forma.

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

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

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

8. Bibliografía

8.1 Fuentes Bibliográficas

[1] Modern Information Retrieval
R.Baeza-Yates, B. Ribeiro-Nieto
Addison- Wesley, 1999

[2] Speech and Language Processing
D.Junafsky, J. H. Martin
Prentice- Hall, 2000

[3] XML Manual de referencia, Heather Williamson
Madrid: McGraw-Hill, 2001

[4] El lenguaje de Unificado de Modelado
Gragy Booch, James Rumbaugh, Ivar Jacobson
Addison- Wesley

[5] Java manual de referencia
Patrick Naughton, Herbert Schildt
McGraw-Hill

[6] Lucene in Action
Otis Gospodnetic, Erik Hatcher
Manning

[7] Ingeniería del Software. Un enfoque práctico
Roger S.Pressman
McGraw-Hill

[8] LexRank: Graph-based Centrality as Salience in Text Summarization
Günes Erkan, Dragomir R. Radev
Department of EECS
University of Michigan

[9] Building Web Applications with UML
Jim Conallen
Addison-Wesley

[10] Patrones de diseño
Erich Gamma

Addison- Wesley

Data mining: Practical Machine Learning Tools and Techniques with Java Implementations
Machine Learning algorithms in Java – Chapter eight
Morgan Kaufmann Publishers

8.2 Fuentes WEB

Procesamiento del Lenguaje Natural – José María Gómez Hidalgo
Universidad Europea de Madrid - http://www.esi.uem.es/~jmgomez/pln/

IRF Framework
http://www.itl.nist.gov/iaui/894.02/projects/irf/irf.html

Lucene: Herramienta que permite tanto indexación como búsqueda de documentos
http://lucene.apache.org/java/docs/
http://www.darksleep.com/lucene/

Kea: Herramienta de extracción de palabras clave
http://www.nzdl.org/Kea/

Weka: Entorno de aprendizaje automático
http://www.cs.waikato.ac.nz/ml/weka/

LexRank: Herramienta de resúmenes
http://tangra.si.umich.edu/~radev/lexrank/lexrank.pdf
http://tangra.si.umich.edu/clair/lexrank/

Pertinence: Herramienta de producción de resúmenes entre otros
http://www.pertinence.net/ps/index.jsp?ui.lang=fr.

Lemur: Herramienta de recuperación
http://newhaven.lti.cs.cmu.edu/

Indri: Herramienta de recuperación e indexación
http://newhaven.lti.cs.cmu.edu/

Experiencias del Grupo COLE en la aplicación de técnicas de Procesamiento del Lenguaje Natural a la Recuperación de Información en español – Miguel A. Alonso, Jesús Vilares, Francisco J. Ribadas
http://coleweb.dc.fi.udc.es/cole/library/ps/AloVilRib2004a.pdf

Etiquetación Robusta del Lenguaje Natural: Preprocesamiento y Segmentación – Jorge Graña Gil, Fco Mario Barcala Rodríguez y Jesús Vilares Ferro
http://coleweb.dc.fi.udc.es/cole/library/ps/GraBarVil2001.ps.gz

[11] Una aplicación RI basada en PLN: el proyecto Erial – Fco Mario Barcala, Eva Mª Domínguez, Miguel A. Alonso, David Cabrero, Jorge Graña, Jesús Vilares, Manuel Vilares, Guillermo Rojo, Mª Paula Santalla y Susana Sotelo – JOTRI 2002
http://coleweb.dc.fi.udc.es/cole/library/ps/ BarDomAloCabGraVilVilRojSanSot2002a.ps.gz

A comparative Study on Feature Selection in Text categorization – Yiming Yang, Jan O. Petersen
http://citeseer.ist.psu.edu/cache/papers/cs/1982/http:zSzzSzwww.cs.cmu.eduzSz~yimingzSzpapers.yyzSzml97.pdf/yang97comparative.pdf

Weka Explorer User Guide for Version 3-3-4 – Richard Kirkby

Tutorial Weka en español
http://www.printsudoku.com/metaemotion/diego.garcia.morate/download/weka.pdf

The Lucene Search Engine
http://www.javaranch.com/newsletter/200404/Lucene.html

Colección textos reuters
http://daviddlewis.com/resources/testcollections/reuters21578/

MANUAL DE USUARIO

1. Introducción

Este manual está dirigido a todas aquellas personas interesadas en el uso de esta herramienta, con el objetivo de guiarles en todos los pasos que son necesarios para su correcta instalación en un equipo. Este proyecto tiene como objetivo proporcionar un marco de trabajo para intentar integrar diversas técnicas y métodos disponibles para el acceso a información textual de forma consistente. De este modo, se podrá simplificar el desarrollo de aplicaciones complejas para al acceso a contenidos textuales, en las que se incorporen diferentes técnicas de recuperación de información.

Además debe de permitir que cualquier usuario de esta arquitectura sea capaz de dotar a una aplicación de todos aquellos servicios que se consideren necesarios. Para ello, se va a describir detalladamente las diferentes formas en las que cualquier usuario puede usar la arquitectura que incorpora un subsistema de adquisición y representación de documentos que permite guardar toda la información extraída en el formato estándar XML.

2. Cómo usar la arquitectura

A continuación se van a indicar los pasos para poder usar la arquitectura, de tal forma que cualquier usuario pueda crear las clases que necesite para poner en funcionamiento el sistema.

2.1 Implementar un Cargador

Para implementar un Cargador es necesario:

- Incluir como primer línea de código, el paquete "*implementacion*"
- Importar el paquete de la arquitectura *arquitex.arquitecture.EstructuraDocumento.**. Ésta incluye la clase "*CapituloStructure*", "*DocumentStructure*", "*ParrafoStructure*", "*RegExpParser*" y "*SubParrafoStructure*".
- Realizar el extends Cargador
- Crear dos constructores: uno sin parámetros y otro con un parámetro que será la ruta del documento original cuya información se quiere extraer.
- Incluir el método "*crearLaEstructuraDelDocumento*" pasándole como parámetros:
  - La información del documento original convertida a String
  - Un String que indique la ruta del documento original
  - El String hash que es el identificador generado por la arquitectura.

  Lo primero que hay que realizar en este método es asignar la ruta al DocumentStructure mediante *asignarElementosRuta*.
  Será necesario dividir la información en este método en capítulos usando CapituloStructure, de tal forma que en cada uno de ellos, su información será extraída en el método "*crearLaEstructuraDelCapitulo*".
  La información que tiene que devolver este método es un objeto de tipo DocumentStructure.
- Incluir el método "*crearLaEstructuraDelCapitulo*", pasándole como parámetro un HashMap que contendrá la información con respecto a un capitulo concreto.
  Este método deberá de devolver un objeto de tipo CapituloStructure.

- Incluir el método "*crearLaEstructuraDelParrafo*", pasándole como parámetro un String que contendrá la información del texto asociado al parrafo en cuestión.
  Este método deberá de devolver un objeto de tipo ParrafoStructure.

- Incluir el método "*crearLaEstructuraDelSubParrafo*", pasándole como parámetro un String que contendrá el titulo del subpárrafo y una lista de HashMap que serán los diversos puntos del que estará compuesto el subparrafo.
  Este método deberá de devolver un objeto de tipo SubParrafoStructure.

- Finalmente es necesario incluir el método "*CargarDocument*" cuya funcionalidad convertir a String el documento original. En esta clase es donde se podrían trasformar determinados aspectos como pueden ser eliminar dobles espacios en blanco, etc.

A continuación se muestra un ejemplo con partes del código del cargador TXT creado.

```
package implementacion;

import arquitex.arquitecture.EstructuraDocumento.*;

public class ConvertTXT2XML extends Cargador {
  public ConvertTXT2XML(){
    super();
  }
  public ConvertTXT2XML(String ruta) {
    super(ruta);
  }

  protected DocumentStructure crearLaEstructuraDelDocumento(String
  documento, String ruta, String h) {
    DocumentStructure dS=new DocumentStructure();
    dS.asignarElementosRuta(ruta);
    dS.setHash(h);
    StringTokenizer s=new StringTokenizer(documento,"\n");
    String capitulo="";
    HashMap ha=new HashMap();
    while(s.hasMoreTokens()){
        //...
        CapituloStructure cap=this.crearLaEstructuraDelCapitulo(ha);
        dS.addCapitulo(cap);
        ha.clear();
        capitulo="";
        //...
    }
    //...
    return dS;
  }

  protected CapituloStructure crearLaEstructuraDelCapitulo(HashMap map) {
```

```
    CapituloStructure cS=new CapituloStructure();
    String titulocap="";
    if(map.containsKey("titcap"))titulocap=(String)map.get("titcap");
    cS.setTituloCapitulo(titulocap);
    String capitulo=(String)map.get("capitulo");
    StringTokenizer s1=new StringTokenizer(capitulo,"\n");
    while(s1.hasMoreTokens()){
        //...
        ParrafoStructure ps=this.crearLaEstructuraDelParrafo(frase);
        cS.addParrafo(ps);
        //...
    }
    return cS;
  }

  protected ParrafoStructure crearLaEstructuraDelParrafo(String s) {
    ParrafoStructure p=new ParrafoStructure();
    p.setTexto(s);
    return p;
  }

  protected SubParrafoStructure crearLaEstructuraDelSubParrafo(String tit,
  HashMap[] subparrafo) {
    SubParrafoStructure sp=new SubParrafoStructure();
    sp.setSubTitulo(tit);
    for(int i=0;i<subparrafo.length;i++){
      String s=(String)subparrafo[i].get("subtexto");
      sp.addSubTexto(s);
    }
    return sp;
  }

  protected String CargarDocument() {
    String salida="";
    try{
      File source=new File(this.getRutaInicialDocumento());
      FileReader fr=new FileReader(source);
      BufferedReader in=new BufferedReader(fr);
      boolean eof=false;
      do{
        //...
      }
      while(!eof);
      in.close();
    }
    catch(IOException e){System.out.println("Error - "+ e.toString());}
    return salida;
  }
}
```

2.2 Implementar un Servicio sobre documentos individuales

Para implementar un Servicio sobre documentos individuales es necesario:

- Incluir como primera línea de código, el paquete "*implementacion*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.decorador.**. Ésta incluye la clase "*DecoradorDocumento*", "*DecoradorTraductor*", "*DecoradorResumidor*", "*DecoradorExtractor*", "*DecoradorClasificador*", "*DocumentoBase*", "*DocumentoXML*" y "*DecoracionStructure*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.MyException y arquitex.arquitecture.Configuracion,* siendo la primera una clase implementada para las Excepciones y la segunda la encargada de recuperar los parámetros del fichero de configuración.
- Importar, si es necesario, las clases específicas utilizadas por la implementación concreta del servicio sobre documentos individuales considerado. En el ejemplo que se va a ver se importa *net.sf.classifier4J.summariser.SimpleSummariser*
- Dependiendo del servicio que se esté implementando, es necesario realizar la extensión de la superclase correspondiente al tipo de servicio que se desea implementar: "*DecoradorTraductor*", "*DecoradorClasificador*", "*DecoradorExtractor*" o a "*DecoradorResumidor*".
- Crear tres constructores: uno con un parámetro que es el nombre del servicio; otro con dos parámetros que son:
  - Un DocumentoXML sobre el que se va a realizar el servicio
  - El nombre del servicio;

  El último es un constructor de dos parámetros que son:
  - El nombre del servicio
  - La ruta relativa del fichero sobre el que queremos aplicar el servicio.
- Incluir el método "*TecnicaResumen*" si procede de un "*DecoradorResumidor*", "*TecnicaTraduccion*" si procede de "*DecoradorTraductor*", "*TecnicaClasificacion*" si procede de un "*DecoradorClasificacion*" o "*TecnicaExtraccion*" si procede de un "*DecoradorExtractor*" pasándole como parámetro el String sobre el que se va a realizar el servicio
- Incluir el método "*DoctoString*", en el que se va a realizar las modificaciones del objeto DecoracionStructure heredado de la clase

DecoradorDocumento, concretamente se realizan modificaciones sobre el atributo Texto del mismo.
Este método deberá de devolver el texto modificado de la decoración, es decir, un String.

- Incluir el método "*cargarParametros*", pasándole como parámetro un el nombre del servicio. En el se accederá al fichero de configuración para extraer los parámetros específicos de cada una de las herramientas incorporadas.

- Incluir el método "*obtenetParametros*", pasándole como parámetro el nombre del servicio. En el se deberá de implementar una lista de "*Parámetros*", es decir, el nombre del parámetro que consta en el fichero de configuración y el valor. Se devolverá esa lista.

- Incluir el método "p*arametrosResumidor*" en el caso de un "*DecoradorResumidor*"; "p*arametrosTraductor*" en el caso de un "*DecoradorTraductor*"; "p*arametrosClasificador*" en el caso de un "*DecoradorClasificador*" o "p*arametrosExtractor*" en el caso de un "*DecoradorExtractor*", pasándole como parámetro el nombre del servicio. En el se deberá de devolver una lista de los nombres de los parámetros tal y como se encuentran en el fichero de configuración.

- Incluir el método "*TextoConsulta*" pasándole como parámetro la consulta y el nombre del servicio. En él se deberá de devolver el resultado de aplicar el decorador a la consulta. Por ejemplo, si poseemos un índice de documentos en francés que proceden de documentos en español, y realizamos la consulta en español, el sistema debe de traducir esa consulta al francés, para poder lanzarla contra el índice construido sobre los textos en francés.

- Incluir el método "*comprobarParametrosDecorador*" pasándole como parámetro el nombre del servicio. En él se deberá de implementar una función que compruebe si los parámetros que se encuentran en el fichero de configuración son correctos o no.

A continuación se muestra un ejemplo con partes del código del resumidor Classifier4j creado.

```
package implementacion;

import arquitex.arquitecture.Configuracion;
import arquitex.arquitecture.MyException;
import arquitex.arquitecture.decorador.*;
import net.sf.classifier4J.summariser.SimpleSummariser;

public class ResumidorClassifier4j extends DecoradorResumidor{
  protected int lineaParametro;
  public ResumidorClassifier4j(DocumentoXML base,String nombre) throws
  MyException {
```

```
    super(base,nombre);
  }
  public ResumidorClassifier4j(String fichero,String vision) throws Exception{
    super(fichero,vision);
  }
  public ResumidorClassifier4j(String vision) throws Exception{
    super(vision);
  }
  public String TecnicaResumen(String texto) {
    try{
      SimpleSummariser summariser=new SimpleSummariser();
      String result = summariser.summarise(texto,this.lineaParametro);
      return result;
    }
    catch(Exception e){System.out.println("tenemos un problema");}
  }
  public String DocToString() {
    DecoracionStructure ds=this.getResultado();
    return ds.getTextoDecoracion();
  }
  protected void cargarParametros(String nbvision) throws MyException{
    String linea=Configuracion.getPropiedad(nbvision+".lineas");
    this.lineaParametro=Integer.parseInt(linea);
  }
  public List obtenerParametros(String nbvision) throws MyException {
    List l=new ArrayList();
    this.cargarParametros(nbvision);
    int i=this.lineaParametro;
    String a=String.valueOf(i);
    Parametros pd=new Parametros("lineas",a);
    l.add(pd);
    return l;
  }
  public List parametrosResumidor(String nbvision){
    List l=new ArrayList();
    l.add("lineas");
    return l;
  }
  public String TextoConsulta(String a,String nbvision) throws MyException {
    this.cargarParametros(nbvision);
    return this.TecnicaResumen(a);
  }
  public boolean comprobarParametrosDecorador(String nbvision) throws
  MyException {
    String linea=Configuracion.getPropiedad(nbvision+".lineas");
    try{
      int l=Integer.parseInt(linea);
      if(l>=1)return true;
      else return false;
    }
    catch(NumberFormatException e){
      return false;
    }
  }
}
```

2.3 Implementar un Servicio de grupos de documentos

Para implementar un Servicio de grupos de documentos es necesario:

- Incluir como primera línea de código, el paquete "*implementacion*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.decorador.**. Ésta incluye la clase "*DecoradorDocumento*", "*DecoradorTraductor*", "*DecoradorResumidor*", "*DecoradorExtractor*", "*DecoradorClasificador*", "*DocumentoBase*", "*DocumentoXML*" y "*DecoracionStructure*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.Grupos.**. Ésta incluye la clase "*Servicios*", "*ServicioIndexador*", "*ServicioClustering*", "*BusquedaStructure*" y "*BusquedaStructureResult*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.MyException y arquitex.arquitecture.Configuracion,* siendo la primera de ellas una clase implementada para la gestión de Excepciones y la segunda la empleada para recuperar los parámetros del fichero de configuración.
- Importar, si es necesario, las clases específicas requeridas para la implementación concreta del servicio. En el ejemplo que se va a ver se importa *org.apache.lucene.**, para incorporar el API de indexación de Lucene.
- Crear tres constructores: uno sin un parámetros; uno con un parametro que es la ruta donde la implementación concreta del servicio almacenará sus estructuras; uno con tres parámetros que son:
  - Una lista de documentos que se quieren añadir a la estructura
  - El nombre del servicio;
  - La ruta donde almacena sus estructuras

  El último es un constructor de tres parámetros que son:
  - El nombre del servicio
  - El valor que indica si ese servicio es temporal o no
  - La ruta donde se almacenan sus estructuras.
- Incluir el método "*crearIndice*" si se trata de un "*ServicioIndexador*" o "*crearCluster*" si se trata de "*ServicioClustering*". Estos son los métodos encargados de crear las estructuras necesarias para el servicio, dependiendo de la herramienta empleada.
- Incluir el método "*indexarNuevosDocumentos*" si se trata de "*ServicioIndexador*" o "*clusterNuevosDocumentos*" si se trata de "*ServicioClustering*", en el que se le pasa un parámetro que será la lista de documentos XML que se van a añadir al servicio

- Incluir el método "*indexarNuevoDocumento*" si se trata de "*ServicioIndexador*" o "*clusterNuevoDocumento*" si se trata de "*ServicioClustering*", en el que se le pasa un parámetro que será el documentos XML que se va a añadir al servicio.

- Incluir el método "*eliminarDocumentoIndice*" si se trata de "*ServicioIndexador*" o "*eliminarDocumentoCluster*" si se trata de "*ServicioClustering*", en el que se le pasa un parámetro que será el documento XML que se va a eliminar del servicio.

- Incluir el método "*eliminarConjuntoDocumentosIndice*" si se trata de "*ServicioIndexador*" o "*eliminarConjuntoDocumentosCluster*" si se trata de "*ServicioClustering*", en el que se le pasa un parámetro que es la lista de documentos XML que se van a eliminar del servicio.

- Incluir el método "*hacerBusqueda*", en el que se le pasa un parámetro que es la consulta que se quiere realizar y el nombre del servicio. Se deberá de devolver un objeto de tipo BusquedaStructureResult que consiste en una lista de listas. En el caso de ser un indexador, cada una de las posiciones de la lista será el nombre del fichero y la lista asociada será nula. En cambio, si se trata de un clustering, cada posisicón de la lista será un cluster, o grupo de documentos, y las listas que lleve asociadas serán listas de nombres de ficheros.

- Incluir el método "*hacerBusqueda*", en el que se le pasa una BusquedaStructureResult, la consulta y el nombre del servicio. Se realiza la consulta sobre el servicio a partir de los documentos que se encuentren en el objeto BusquedaResultStructure que se le pase. Éste deberá de devolver un objeto de tipo BusquedaStructureResult.

- Incluir el método "*cargarParametros*", pasándole como parámetro un el nombre del servicio. En él se accederá al fichero de configuración para extraer los parámetros de cada una de las herramientas por separado.

- Incluir el método "*obtenerParametros*", pasándole como parámetro el nombre del servicio. En él se deberá de implementar una lista de "*Parámetros*", es decir, el nombre del parámetro que consta en el fichero de configuración y el valor. Se devolverá esa lista.

- Incluir el método "p*arametrosIndexador*" en el caso de un "*ServicioIndexador*" o "p*arametrosCluster*" en el caso de un "*ServicioClustering*", pasándole como parámetro el nombre del servicio. En él se deberá de devolver una lista de los nombres de los parámetros tal y como se encuentran en el fichero de configuración.

- Incluir el método "*comprobarParametrosServicios*" pasándole como parámetro el nombre del servicio. En él se deberá de implementar una función que devuelva si los parámetros que se encuentran en el fichero de configuración son correctos o no.

A continuación se muestra un ejemplo del código que implementa el servicio de indexación basado en Lucene utilizado en nuestra arquitectura:

```
import arquitex.arquitecture.Configuracion;
import arquitex.arquitecture.MyException;
import arquitex.arquitecture.Grupos.*;
import arquitex.arquitecture.decorador.*;

public class IndexadorLucene extends ServicioIndexador{
  protected String nbanalizador;
  private Analyzer analizador;
  public IndexadorLucene(){
    super();
  }
  public IndexadorLucene(ArrayList elem,String nbi,String path) {
    super(elem,nbi,path);
  }
  public IndexadorLucene(String path) throws MyException{
    super(path);
  }
  public IndexadorLucene(String nbvision,String temporal,String path) throws
  MyException{
    super(nbvision,temporal,path);
  }
  public void crearIndice() throws Exception {
    Directory dir=FSDirectory.getDirectory(this.getPathIndexador(),true);
    IndexWriter writer= new IndexWriter(dir, null,true);
    writer.close();
  }

  public void indexarNuevosDocumentos(List docs) throws Exception {
    Iterator it=docs.iterator();
    while(it.hasNext()){
      DocumentoXML doc=(DocumentoXML)it.next();
      this.indexarNuevoDocumento(doc);
    }
  }
  public void indexarNuevoDocumento(DocumentoXML doc) throws Exception {
    Directory dir=FSDirectory.getDirectory(this.getPathIndexador(), false);
    IndexWriter writer=new IndexWriter(dir,this.analizador,false);
    IndexReader reader=IndexReader.open(dir);
    int cont=0;
    for(int i=0;i<reader.numDocs();i++){
      if(reader.document(i).get("id").equals(doc.getHashId())){
        cont=1;
        break;
      }
    }
    //...
  }

  public void eliminarDocumentoIndice(DocumentoXML doc) throws Exception {
    Directory dir=FSDirectory.getDirectory(this.getPathIndexador(), false);
    IndexReader reader=IndexReader.open(dir);
```

```
    Searcher searcher=new IndexSearcher(dir);
    Query query=QueryParser.parse(doc.getHashId(),"id",this.analizador);
    Hits hits=searcher.search(query);
    for(int i=0;i<hits.length();i++){
      int id=hits.id(i);
      reader.delete(id);
    }
    reader.close();
    //...
  }
  public void eliminarConjuntoDocumentosIndice(List docs) throws Exception {
    Iterator it=docs.iterator();
    while(it.hasNext()){
      DocumentoXML doc=(DocumentoXML)it.next();
      this.eliminarDocumentoIndice(doc);
    } }
  public BusquedaStructureResult hacerBusqueda(String query,String fuente)
  throws MyException{
    BusquedaStructureResult b=new BusquedaStructureResult();
    //...
    return b;
  }
  public BusquedaStructureResult hacerBusqueda(BusquedaStructureResult bs,
  String query,String fuente)throws MyException {
    BusquedaStructureResult b=new BusquedaStructureResult();
    //...
    return b;
  }
  protected void cargarParametros(String nbvision) throws MyException{
    try{
      String clase=Configuracion.getPropiedad(nbvision+".claseAnalizadora");
      Class an= Class.forName("org.apache.lucene.analysis."+clase);
      this.analizador = (Analyzer) an.newInstance();
      this.nbanalizador=clase;
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  public List obtenerParametros(String nbvision) throws MyException {//...}
  public List parametrosIndexador(String nbvision) {//...}
  public boolean comprobarParametrosServicio(String n) throws MyException {
    String claseAnalizadora=Configuracion.getPropiedad(n+".claseAnalizadora");
    try {
      Class an= Class.forName("org.apache.lucene.analysis."+claseAnalizadora);
      Analyzer a = (Analyzer) an.newInstance();
      return true;
    } catch (Exception e) {
      return false;
    }
  }
}
```

2.4 Implementar la presentación de los documentos

Para implementar la presentación de los documentos es necesario:

- Incluir como primera línea de código, el paquete "*implementacion*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.decorador.**. Ésta incluye la clase "*DecoradorDocumento*", "*DecoradorTraductor*", "*DecoradorResumidor*", "*DecoradorExtractor*", "*DecoradorClasificador*", "*DocumentoBase*", "*DocumentoXML*" y "*DecoracionStructure*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.MyException y arquitex.arquitecture.Configuracion,* siendo una la clase implementada en la primera la clase implementada para las Excepciones y para recuperar los parámetros del fichero de configuración.
- Importar la clase necesaria para la implementación de la presentación. En el ejemplo que se va a ver se importa *arquitex.VisualizacionDocumentos.VisualizacionHTML.*
- Realizar la extensión de la superclase "*VisualizacionHTML*".
- Crear un constructor sin parámetros
- Incluir el método "*VisualizacionDocBase*", pasándole como parámetro la dirección absoluta del fichero. Este es el método encargado de crear la visualización del documento base.
- Incluir el método "*VisualizacionDocDecorado*", pasándole como parámetro la dirección absoluta del fichero. Este es el método encargado de crear la visualización del documento decorado.
- Incluir el método "*getVisualizacionLista*"
- Incluir el método "*setVisualizacionLista*"

A continuación se muestra un ejemplo de partes del código del código que implementa la clase de la visualizacionHTMLResumen. Es necesario indicar que en el momento de realizar las llamadas a la superclase en VisualizacionDocBase y VisualizacionDocDecorado, se obtiene la presentación por defecto de la arquitectura. Lo mismo ocurriría si se pusiera en el método getVistaLista una llamada a la superclase:

```
package implementacion;

import arquitex.VisualizacionDocumentos.VisualizacionHTML;
import arquitex.arquitecture.Configuracion;
import arquitex.arquitecture.MyException;
import arquitex.arquitecture.decorador.*;
```

```
public class VisualizacionHTMLResumen extends VisualizacionHTML{
  public VisualizacionHTMLResumen(){
    super();
  }
  public void VisualizacionDocBase(String fichero) throws MyException {
    super.VisualizacionDocBase(fichero);
  }

  public void VisualizacionDocDecorado(String fichero) throws MyException {
    super.VisualizacionDocDecorado(fichero);
  }

  public String getVisualizacionLista(String s, DocumentoXML d,String
  visionllamante) {
    String html1="";
    if(d instanceof DecoradorResumidor){
      DecoradorResumidor dr=(DecoradorResumidor)d;
      DecoracionStructure ds=dr.getResultado();
      String nombre=ds.getNombreDecoracion();
      String dir;
      //...
      try {
        String fuente=Configuracion.getPropiedad(visionllamante+".fuente");
        if(fuente.equals("DocumentoBase"))
          html1="<br>  <img  src=\"./imagenes/flecha_azul.jpg\">     <a
          href=\"./presentacion.jsp? d=/DocBase/"+s+"\" class=\"enlace\">";
        else
          html1="<br>  <img  src=\"./imagenes/flecha_azul.jpg\">     <a
          href=\"./presentacion.jsp?d=/DocumentoDecorado/"+nombre+"/"+s+"\"
          class=\"enlace\">";
        html1+=d.getDocument().getTitulo()+"</a> ";
        //...
        html1+="<span class=\"texto\">El resumen de la vision mediante "+nombre+"
        es: </span><br>";
        html1+="<span class=\"texto\">"+texto2+"... </span>";
      } catch (Exception e1) {
        e1.printStackTrace();
      }
    }
    return html1;
  }

  public void setVisualizacionLista(String h) {
    super.setVisualizacionLista(h);
  }
}
```

2.5 Implementar la presentación de los resultados de las búsquedas de documentos mediante un servicio de grupo de documentos

Para implementar la presentación de los documentos es necesario:

- Incluir como primera línea de código, el paquete "*implementacion*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.Grupos.**. Ésta incluye la clase "*Servicios*", "*ServicioIndexador*", "*ServicioClustering*", "*BusquedaStructure*" y "*BusquedaStructureResult*".
- Importar el paquete de la arquitectura *arquitex.arquitecture.MyException y arquitex.arquitecture.Configuracion,* donde la primera clase da soporte para las Excepciones y la segunda se encarga de recuperar los parámetros del fichero de configuración.
- Importar las clases necesarias para la implementación específica de la presentación. En el ejemplo que se va a mostrar se importa *arquitex.VisualizacionDocumentos.VisualizacionHTML.*
- Realizar el extends a "*VisualizacionServHTML*".
- Crear un constructor sin parámetros
- Incluir el método "*getVistaLista*"

```
package implementacion;

import arquitex.arquitecture.MyException;
import arquitex.arquitecture.Grupos.*;
import arquitex.VisualizacionServicios.*;

public class VisualizacionServHTMLConcreto extends VisualizacionServHTML{
  public VisualizacionServHTMLConcreto(){
    super();
  }
  public String getVistaLista(BusquedaStructureResult b, String vision) throws
  MyException {
    return super.getVistaLista(b,vision);
  }
}
```

3. Como usar la aplicación

Antes de comenzar a explicar todos los pasos necesarios para la instalación de la aplicación, se detallan los requisitos, tanto hardware como software, mínimos que ha de cumplir el equipo en el que vaya a ser instalado, con el fin de garantizar su correcto y eficiente funcionamiento.

3.1 Requisitos Hardware

Las prestaciones mínimas del equipo en el que se desee instalar y ejecutar han de ser las siguientes:

- Requisitos Hardware
  - CPU 400 MHz o superior (recomendable 1.5GHz). El tipo de procesador es indiferente. Lo único necesario es que exista una distribución de la Máquina Virtual Java para dicho procesador.
  - Memoria 512 MB (recomendable 1GB).
  - Disco Duro de gran capacidad (recomendable superior a 20 GB).
  - Conexión a Internet, para permitir a la herramienta Google realizar las traducciones.
  - Unidad de CD-ROM

3.2 Requisitos -Software

El equipo en el que se desee instalar la aplicación debe contar con:

- Requerimientos Software del Servidor
  - Sistema operativo Linux, para exista una Máquina Virtual Java y se pueda ejecutar la herramienta de extracción de palabras clave Erial. En el caso de no usar la herramienta Erial, se podría usar el sistema operativo Windows XP.
  - Máquina Virtual Java JDK 1.4.2 (http://java.sun.com).
  - Servidor de páginas Web y contenedor de Servlets Jakarta Tomcat 5.0.
  - Sistema Erial: sistema de extracción y recuperación de información mediante análisis lingüístico.
  - Librería Classifier4j-0.6.jar: sistema de clasificación y resumidor
  - Librería Lucene1.4.3.jar: sistema de indexación

- Librería weka.jar: sistema de clasificación y clustering.

- Requerimientos Software del Cliente
  - Navegador de Internet, recomendado Mozilla Firefox 1.7.10. En caso de usar el sistema operativo Windows, también se puede usar el navegador Internet Explorer 6.0 o superior.

Para facilitar la instalación de los componentes software de libre distribución, tanto el JDK como el Tomcat están incluidos en el CD que se adjunta con la documentación.

3.3 Instalación y ejecución

Para la correcta instalación y posterior ejecución de la aplicación es necesario llevar a cabo los siguientes pasos:

3.3.1 Instalar Java Development Kit 1.4

Para que funcione correctamente Tomcat es obligatorio que previamente el sistema en el que va a funcionar tenga ya instalado y correctamente configurado, el JDK de Sun (que también se incluye en el CD-ROM).

- **Windows:** j2sdk-1_4_2_09-windows-i586-p.exe
- **Linux:** j2sdk-1_4_2_09-linux-i586-p.bin

El JDK, se puede descargar desde el sitio web de Sun que está en: http://java.sun.com/j2se/

El usuario deberá colocarse en el directorio donde se encuentra el archivo.

3.3.1.1 Trabajar en Windows

El JDK requiere configurarse con dos variables de entorno para su correcta operación :

- **PATH** : Define la ruta de acceso para los binarios del sistema; la modificación de esta variable permite acceder los ejecutables Java proporcionados con el JDK de cualquier directorio
- **CLASSPATH**: Define las rutas de acceso para las diversas librerías empleadas en ambientes Java; su modificación será descrita a lo largo del curso

Para definir la variable PATH:

Seleccione Inicio - MiPC y con el botón derecho Propiedades – Opciones avanzadas y variable de entrono. Localice "Path" en las variables del usuario y variables de Sistema. Debe agregar la ruta de acceso para el JDK al final de esta variable, un valor típico es el siguiente :

C:\j2sdk1.4.2_<numero_version>\bin

3.3.1.2 En el caso de trabajar en Linux

Es necesario escribir en algunos directorios que normalmente están restringidos al usuario. Para ello hay que acceder al sistema como usuario "root" (el administrador del sistema) para ello ejecute:

su – :solicitando la contraseña correspondiente a root.

A continuación se deben cambiar los permisos del archivo para poderlo ejecutar, con la siguiente instrucción:

chmod 755 j2sdk-1_4_2_09-linux-i586.bin

Ahora instale la máquina virtual java, con la instrucción:

./j2sdk-1_4_2_09-linux-i586.bin

Al ejecutar el programa, éste pregunta si el usuario está de acuerdo con los términos de la licencia. Presione la tecla “enter” para aceptar.

El programa, al ser ejecutado, crea un directorio donde está la versión de java que vamos a instalar.

Se observa que el nombre del directorio es muy largo y difícil de recordar, por lo que se recomienda cambiarlo por uno más sencillo como se indica a continuación:

mv j2sdk-1_4_2_09-linux-i586 java

El siguiente paso es mover el archivo al directorio donde se desea hacer la instalación.

Ejemplo:
mv java /usr/local

En el ejemplo anterior, se colocaron los binarios de java en un directorio donde los usuarios pueden ejecutarlo. Ahora es necesario configurar la variable de entrono PATH, agregando 3 líneas al final del archivo /etc/profile, para que sea válido para todos los usuarios del sistema de la siguiente manera:

vi /etc/profile

Y las líneas que se deben de agregar al final del archivo son:

export JAVA_HOME=/usr/local/java
export JAVA_BIN=/usr/local/java/bin
export JAVA_OPTS= -Xmx512M
export PATH=$PATH:$JAVA_HOME:$JAVA_BIN

Para que los cambios tengan efecto se debe hacer que el procesador los lea mediante la siguiente instrucción:

source /etc/profile

Finalmente se prueba que java esté correctamente instalado, ejecutándolo de la siguiente forma:

java

Java debe responder y dar datos acerca de su configuración, su versión, etc. En caso de devolver un error revise cada una de las instrucciones y vuelva a probar. Es necesario que este paso esté correcto para poder comenzar con la instalación del servidor jakarta-tomcat.

3.3.2 Instalar y ejecutar el servidor de páginas Web

Si Tomcat no se encuentra todavía instalado en nuestro equipo, suponiendo ya instalado el JDK, se debe copiar el directorio el directorio software/Jakarta-tomcat del CD adjunto al disco duro y a continuación se debe de modificar:

- **Windows:** Jakarta-tomcat-5.0.28.exe
- **Linux:** Jakarta-tomcat-5.0.28.tar.gz

3.3.2.1 En el caso de trabajar en Windows

Instalar Tomcat en Windows es muy sencillo usando el instalador. Su interfaz y funcionalidades son similares a otros instaladores.

El instalador creará le guiará durante el proceso de instalación. Siga las instrucciones y obtendrá una instalación completa del servidor.

Sólo resta verificar que el servidor de jakarta esté funcionando correctamente, para lo cual, se debe teclear la siguiente dirección URL en el navegador de web:

http://localhost:8080/

3.3.2.2 En el caso de trabajar en Linux

Para desempaquetar el archivo jakarta-tomcat-5.0.28.tar.gz, es recomendable que el usuario se coloque en el directorio donde esté situado el archivo.

Ahora se debe desempaquetar y descomprimir el archivo con la siguiente instrucción:

tar –zxvf jakarta-tomcat-5.0.28tar.gz

Esto crea un directorio llamado jakarta-tomcat-5.0.28 el cual contiene todos los archivos de tomcat. En el ejemplo anterior se observa que el nombre del directorio es muy largo y difícil de recordar, por lo que se recomienda cambiarlo por uno más sencillo. Convencionalmente se le pone el nombre de jakarta-tomcat con la siguiente instrucción:

mv jakarta-tomcat-4.1.24 jakarta-tomcat

Ahora se pueden cambiar los archivos a un directorio donde puedan ser ejecutados por todos los usuarios, como por ejemplo /usr/local. Este paso se puede hacer de la siguiente manera:

mv jakarta-tomcat /usr/local

Para que funcione correctamente es necesario declarar una variable de entorno llamada **CATALINA_HOME** que contenga la ruta donde está instalado jakarta-tomcat, y esto se puede lograr editando el archivo /etc/profile de la siguiente manera:

vi /etc/profile

Y agregando la siguiente línea al final del archivo:

export CATALINA_HOME=/usr/local/jakarta-tomcat

Para que el cambio se realice se debe obligar al procesador a que lea el archivo para que las variables de entorno sean creadas. Lo anterior se logra con la siguiente instrucción:

source /etc/profile

Ahora jakarta-tomcat tiene todo lo necesario para poder funcionar correctamente.

Para que se inicie el servicio se debe teclear la siguiente instrucción:

$CATALINA_HOME/bin/startup.sh

Sólo resta verificar que el servidor de jakarta esté funcionando correctamente, para lo cual, se debe teclear la siguiente dirección URL en el navegador de web:

http://localhost:8080/

El número 8080 en la dirección, le indica al navegador que se conecte al puerto 8080 del equipo. Como resultado, se debe observar en el navegador una página que indica que se ha instalado correctamente Jakarta-Tomcat. Dicha página contiene unos pequeños ejemplos que se pueden probar para asegurarse de que el servidor está correctamente instalado.

Si se hiciera algún cambio en el archivo $CATALINA_HOME/conf/server.xml se debe detener el servicio y después se debe volver a iniciar, a fin de que los cambios puedan ser apreciados. Esto se logra con la instrucción:

$CATALINA_HOME/bin/shutdown.sh

En caso de que alguno de estos pasos no funcione correctamente revise los pasos anteriores.

3.3.3 Instalar la aplicación

Una vez instalado la máquina virtual y el servidor Tomcat, el siguiente paso es instalar los archivos correspondientes al sistema. Estos archivos se encuentran en el CD-ROM de instalación.

Copiar la carpeta *arquitex* situada en el directorio *implementación* al directorio *webapps* de Tomcat. Contiene el código del sistema y todo lo necesario para su funcionamiento.

Para comprobar el funcionamiento del servidor, solo es necesario arrancar el servidor Tomcat, abrir un navegador y escribir:

http://nombreDelServidor:8080/arquitex/

Si funciona correctamente, mostrará la página de inicio de la aplicación.

3.4 Funcionamiento de la aplicación

En este apartado, se describe el funcionamiento de la aplicación de ejemplo desarrollada para mostrar, de forma breve, el funcionamiento de la arquitectura implementada. Se desarrolló una interfaz amigable donde se encontraran tódas las funcionalidades de una forma fácil.

3.4.1 Página principal

Nada más arrancar la aplicación, se muestra la página de carga. Esto es debido a que en el momento de entrar, el servidor comprueba, por un lado, la existencia del fichero de configuración, en cuyo caso verifica si éste está correctamente configurado, y verifica que existan todos los servicios sobre documentos individuales, los servicios de grupos de documentos y las presentaciones tanto de los documentos base como de los documentos decorados, todo ello en función del fichero de configuración.

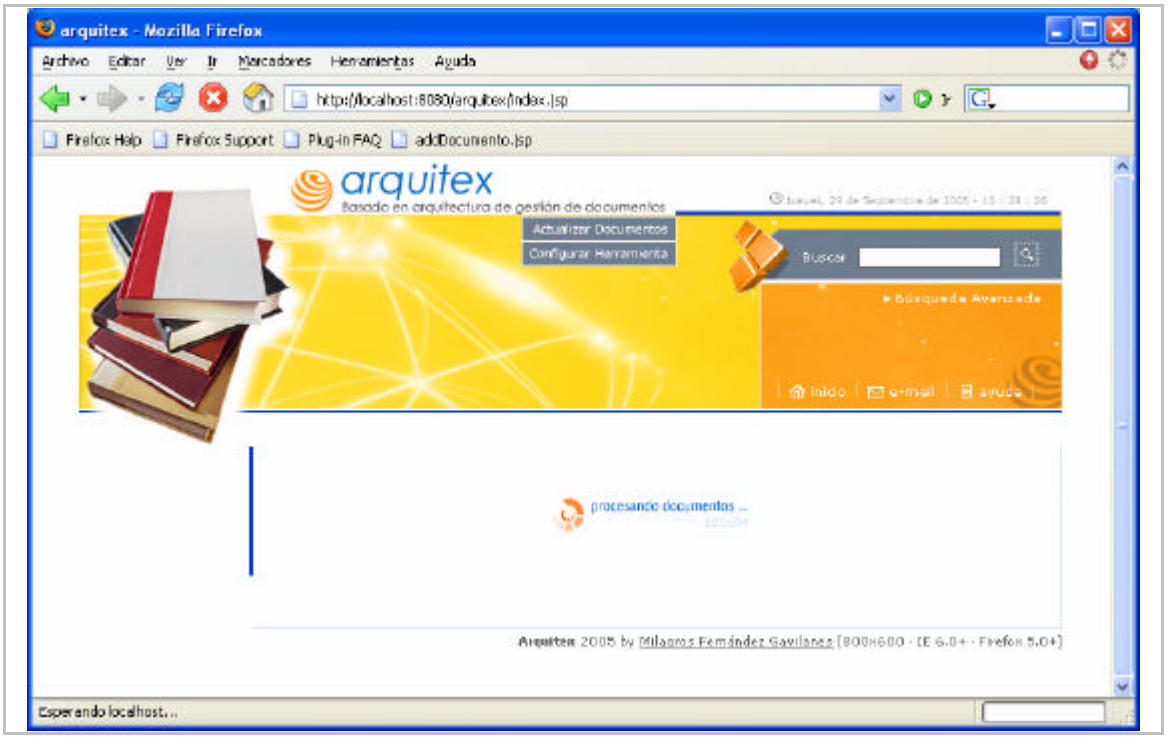


Figura 195: Manual de Usuario: Página de carga

Todo este proceso de carga puede tardar unos segundos dependiendo de la cantidad de documentos que se encuentran en ese momento almacenados y de si es necesario crear algún servicio sobre documentos individuales, servicio de grupo de documentos o alguna presentación. Este es un proceso más o menos rápido en función de la cantidad de documentos que se manejen.

Mientras no se finalice el proceso de carga, no se tiene opción a pulsar ninguno de los botones que se encuentran en la interfaz.

En el momento de finalizar la verificación de los documentos en el servidor, la página de inicio redirecciona a una nueva ventana, como se muestra en la página que viene a continuación con los siguientes elementos.

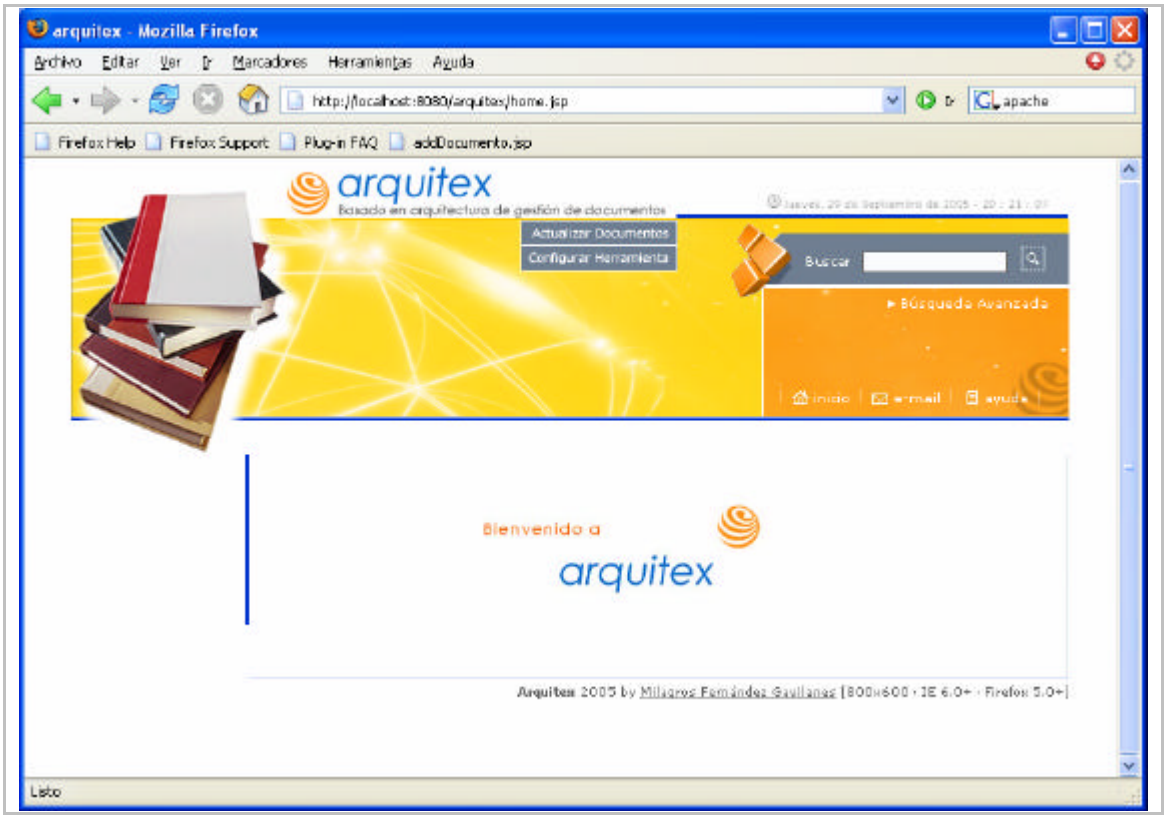
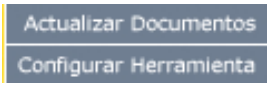


Figura 196: Manual de Usuario: Página de bienvenida

La aplicación **arquitex** cuenta con un menú situado en la parte superior central de la ventana que muestra las funcionalidades típicas que se podrían ofrecer a un administrador de la herramienta:



La primera de las funcionalidades que aquí se muestran es "*Actualizar Documento*s" que permitirá añadir o eliminar documentos de la colección. La segunda opción "*Configurar Herramienta*" permite, como su nombre indica, configurar la herramienta, es decir, cambiar el fichero de configuración, para de ese modo cambiar los servicios asociados a la colección. Ambos se verán con más detalla en los próximos apartados.

El resto de los menús que se encuentran situados en la parte derecha de la ventana, muestran las funcionalidades típicas que se podrían ofrecer a cualquier usuario que necesitase de este tipo de sistema.

Por un lado está:

siendo cada una por separado:

inicio: al pulsar sobre este botón se regresa a la página de inicio

e-mail: en caso de algún tipo de duda, se puede contactar con el webmaster de la aplicación

ayuda: en caso de necesitar de algún tipo de soporte sobre el uso de la herramienta, se puede consultar la ayuda.

Además de estos menús, existen otros como son:

El menú **Buscar** es un acceso directo a cualquiera de los servicios que se tenga en ese momento en el fichero de configuración. Este acceso directo a alguno de los servicios se configura en el menú "*Configurar Herramienta*" que se verá en los siguientes apartados.

El menú **Búsqueda Avanzada** permite tener acceso a una página donde se podrán realizar consultas sobre cualquiera de los servicios que se encuentren en ese momento disponible en el fichero de configuración.

3.4.2 Actualizar los Documentos

Cuando pulsamos sobre el botón de "*Actualizar Documentos*", lo primero que se observa es un pequeño submenú:

La ventana que aparece por defecto al pulsar la opción es "*Añadir Documentos*", es la que se muestra a continuación:

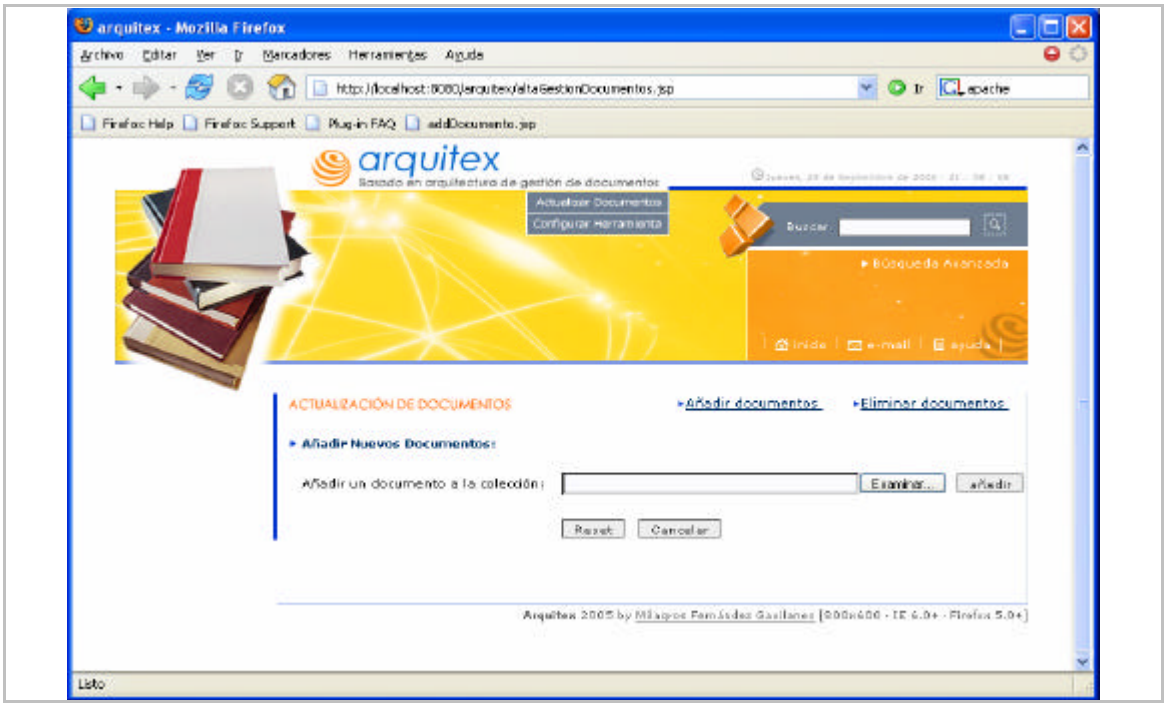


Figura 197: Manual de Usuario: Añadir Documento

En esta ventana se puede añadir un nuevo documento a la colección. Pulsando el botón "*Examinar*" se abre una cuadro de diálogo en el cual se podrá elegir el fichero que desea añadir. Únicamente podrá seleccionar ficheros con extensión: .htm, .pdf, .txt y .doc, en caso contrario se le informará del error.

Figura 198: Manual de Usuario: Ventana de error añadir documento

Una vez seleccionado el fichero pulse el botón "*Añadir*" para incorporarlo a al colección. En ese momento se extraerá la información del documento, se generará el XML asociado al mismo, se crearán las visiones correspondientes y se incorporarán a los servicios de grupos de documentos, finalmente se generan las presentaciones vinculadas a cada uno.

En el caso de seleccionar la otra opción, "*Eliminar documentos*" se podrán eliminar todos los documentos base que se desee. De esta forma, además de eliminar los documentos base, también se eliminarán sus visiones, sus presentaciones junto con los servicios de grupo de documentos en los que esté incluido.

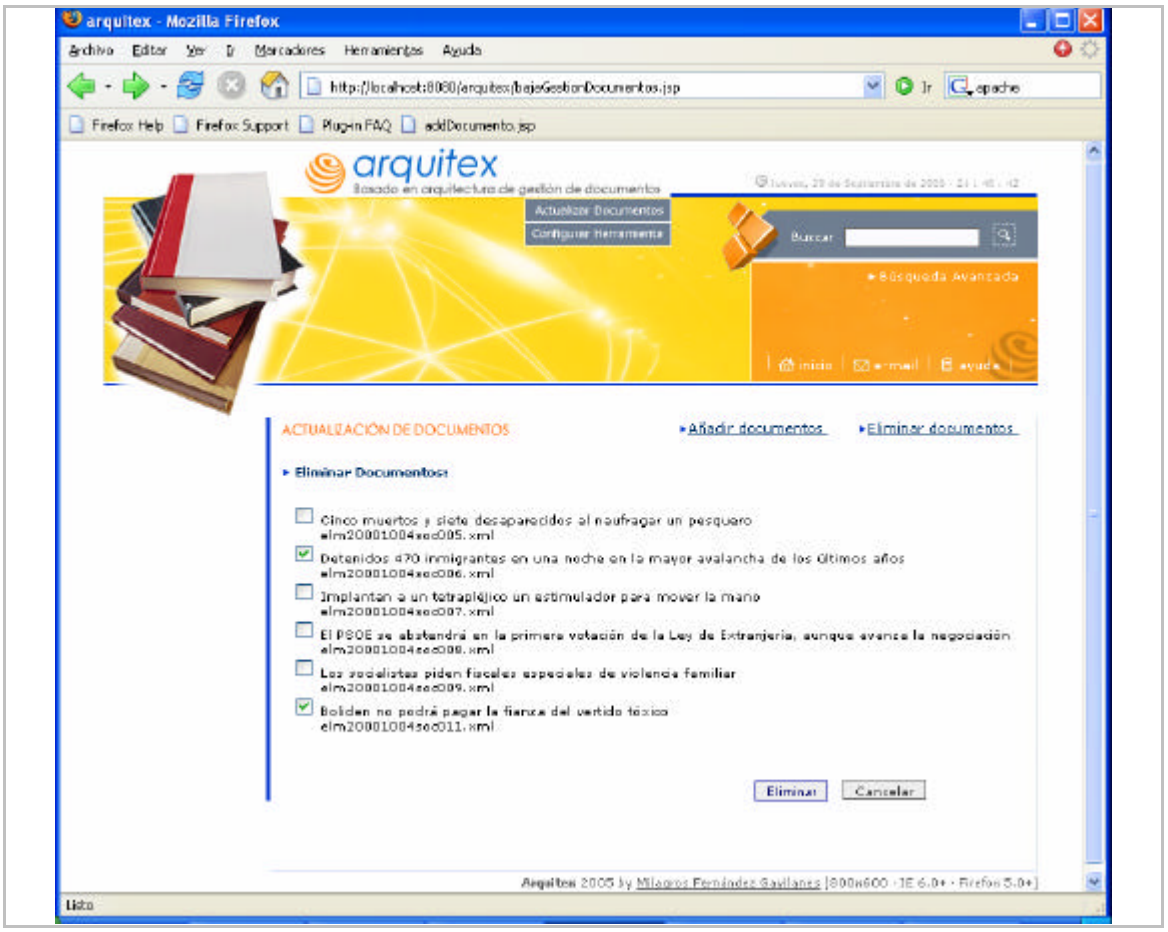


Figura 199: Manual de Usuario: Eliminar Documentos

Hay que indicar que la información que se visualiza sobre cada documento es el titulo así como el nombre del fichero al que hace referencia.

En el momento de pulsar *eliminar*, se procederá a eliminar el documento base asociado así como todas sus visiones y presentaciones. De forma análoga se eliminarán todas las referencias del documento en los servicios de grupos de documentos.

3.4.3 Configurar la Herramienta

Esta opción es la elegida cuando lo que se pretende es *modificar* el fichero de configuración de la arquitectura. De este modo se pueden incluir todos los servicios que se necesiten proporcionar sobre los documentos.

Esta opción consta de varios pasos, que se van a detallar a continuación.

3.4.3.1 Paso 1: establecer directorios, cargadores y nombres de servicios

Figura 200: Manual de Usuario: Configurar Herramienta, ventana 1

El usuario que desee configurar la herramienta va a poder seleccionar:

- Seleccionar el directorio donde quiere almacenar los documentos base
- Seleccionar el directorio donde quiere almacenar los documentos decorados
- Seleccionar el directorio donde quiere almacenar los servicios grupos de documentos

- Seleccionar el directorio donde almacenar las presentaciones de los documentos tanto base como decorados.

Es necesario indicar que dichas rutas van a ser rutas relativas. Ya que el directorio que los va a contener todos está situado en CATALINA_HOME/webapps/arquitex/.

Para evitar que se inserten espacios en blanco en los directorios, la aplicación los elimina.

A continuación se deberán de indicar el nombre de las clases que implementen algún tipo de cargador .doc, .html, .pdf, .txt.

En el caso de que la clase que se inserte no exista o que no extienda la clase apropiada, es decir, si se rellena el campo de .txt y la clase implemente un cargador de .doc, se mostrará el siguiente mensaje de error:

Figura 201: Manual de Usuario: mensaje de error1 Configurar Herramienta

En el caso de que no se inserte ningún cargador en ninguno de los 4 campos, se mostrará el siguiente mensaje:

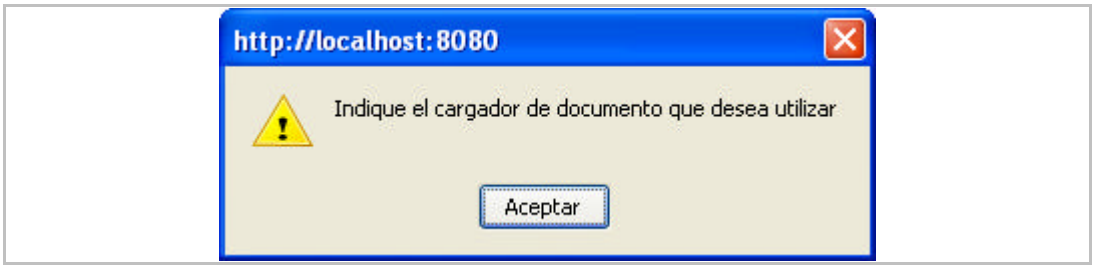


Figura 202: Manual de Usuario: mensaje de error2 Configurar Herramienta

A continuación, es necesario cumplimentar los campos referentes a los servicios sobre documentos individuales. Cada uno de los servicios correspondientes a un determiando grupo se separaran por “,”. En caso de poner espacios blancos, éstos se eliminarán.

Hay que indicar que es obligatorio cubrir alguno de los servicios sobre documentos individuales, sino se mostrará un error del estilo:

Figura 203: Manual de Usuario: mensaje de error3 Configurar Herramienta

También es necesario comentar que no se pueden repetir los nombres de los servicios sobre documentos individuales, es decir, cada uno de ellos debe de poseer un nombre único. En caso contrario el mensaje de error será el siguiente:

Figura 204: Manual de Usuario: mensaje de error4 Configurar Herramienta

El siguiente paso es indicar cuales son los servicios de grupos de documentos. Hay indicar todos los indexadores y todos los clusterings. En este caso ocurre lo mismo que en los servicios sobre documentos individuales, es decir, cada uno de los servicios que se indiquen en un mismo campo, deberá de ir separado por ",". En caso de colocar espacios en blanco, estos serán eliminados.

En el caso de no indicar ningún servicio sobre grupo de documentos, se mostrará un mensaje de error:

Figura 205: Manual de Usuario: mensaje de error4 Configurar Herramienta

En el caso de que alguno de estos servicios posea un nombre repetido con respecto tanto a los servicios sobre documentos individuales como de grupos de documentos, el error será el mismo que el que se muestra en la Figura 204.

El último paso de esta ventana indica si se desea incorporar a la aplicación un sistema de cadena de servicios, es decir, realizar una consulta sobre un servicio de grupos de documentos y el resultado obtenido volver a realizar la misma operación.

Dicho esto, en caso de querer incorporar una cadena de servicios, es necesario indicar el nombre que le queremos dar, siguiendo la premisa de no repetir nombres.

Hay que destacar que en el caso de pulsar "*siguiente*", no se guarda ninguno de los cambios realizados en el formulario. Si se pulsa "*modificar*", se guardan los valores que se acaban de insertar.

3.4.3.2 Paso 2: Indicar los parámetros comunes a los servicios

En el siguiente paso, lo primero que se puede observar es lo siguiente:

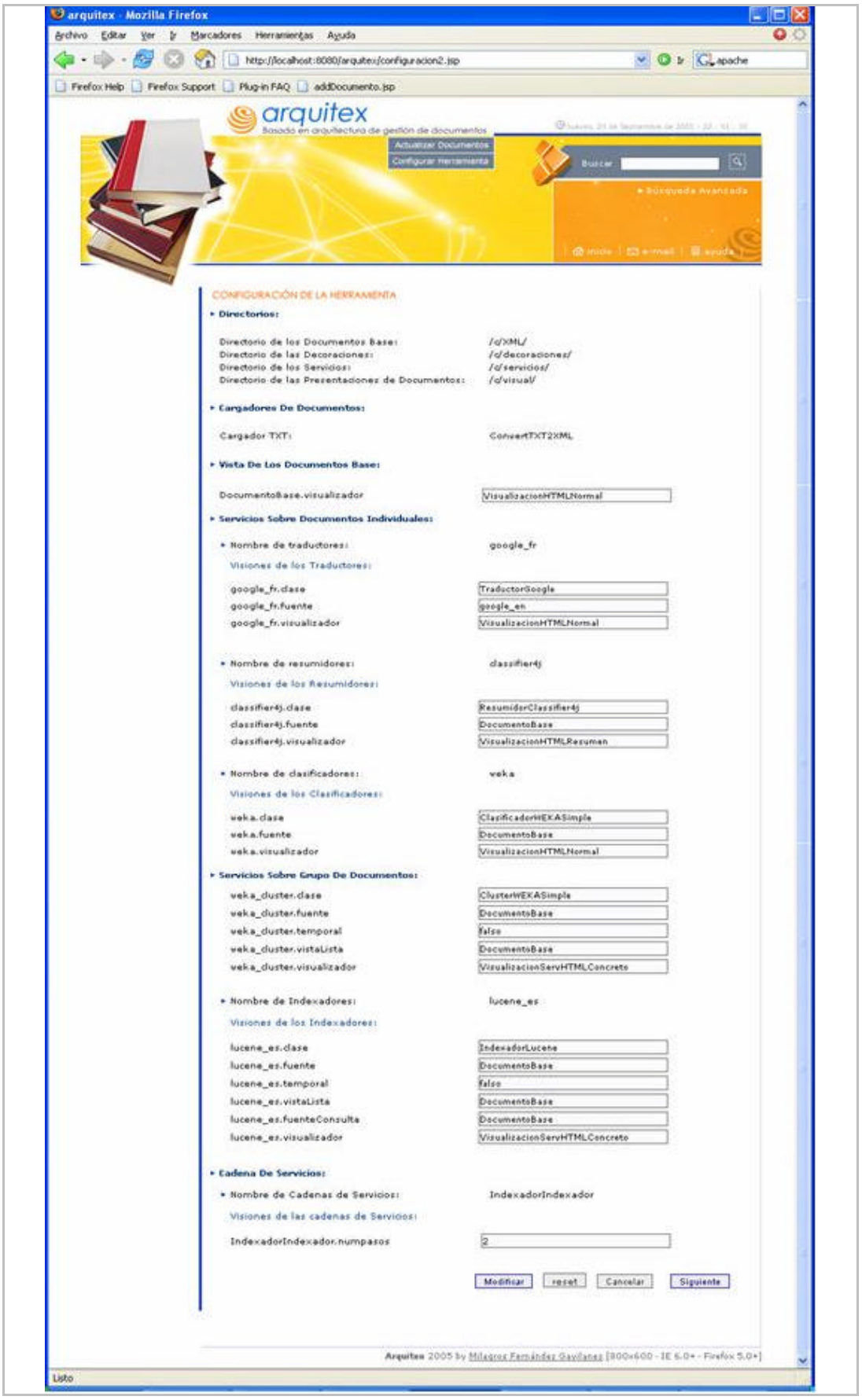


Figura 206: Manual de Usuario: Configurar Herramienta, ventana 2

Lo primero que hay que hacer es indicar cual es la clase que implementa la visualización de los documentos base. En el caso de que ese campo esté vacío o que no sea la clase adecuada, se mostrará un mensaje de error del estilo:

Figura 207: Manual de Usuario: mensaje de error5, Configurar Herramienta

Por cada uno de los servicios sobre documentos individuales, es necesario indicar la clase que lo implemente, el nombre de la fuente sobre la que se va a crear esa decoración y la clase que implementa la visualización de ese servicio.

En el caso de que alguno de los campos de cualquiera de estos servicios sobre documentos individuales esté vacío, se mostrará el mensaje:

Figura 208: Manual de Usuario: mensaje de error6, Configurar Herramienta

Si la clase que se introduce, o bien no es una clase valida, o bien no es una subclase de la clase principal de ese servicio, la aplicación devolverá el error:

Figura 209: Manual de Usuario: mensaje de error7, Configurar Herramienta

La fuente de las visiones tiene que ser o bien un "DocumentoBase" (haciendo referencia a los documentos base) o bien el nombre de una visión que esté creada como podría ser el google_en. En el caso de definir una fuente que no se encuentre, será necesario esperar al último paso de la configuración para determinar su origen.

En el caso de los servicios de grupos de documentos, ocurre exactamente lo mismo que en el caso de los servicios sobre documentos individuales, es decir, es necesario indicar las clases correctas así como las fuentes. Además de ello, se tiene que tener en cuenta que dichos servicios pueden ser creados temporalmente o no, es decir, en el caso de indicar que el servicio tiene la propiedad a *falso*, significará que el servicio se creará una vez; en el caso de indicar este parámetro *verdadero*, lo creará y eliminará cada vez que se realice una consulta sobre el servicio. Por lo tanto los valores de este parámetro solo pueden ser "verdadero" o "falso". En caso contrario, se muestra el mensaje:

Figura 210: Manual de Usuario: mensaje de error8, Configurar Herramienta

También es preciso destacar el parámetro vistaLista de los servicios. Este parámetro indica en que tipo de visualización se va a basar cada resultado de un documento que proporciona un servicio. Si vistaLista es DocumentoBase, significa que la visualización que se va a emplear para cada unos de los resultados que proporciona un servicio va a usar el visualizador de DocumentoBase; si vistaLista fuese classifier4j, la visualización de la lista se basaría en el visualizador que hace

uso classifier4j. Por lo tanto vistaLista tiene que ser el nombre de un servicio sobre documentos individuales o el propio DocumentoBase. En caso contrario, se muestra:

Figura 211: Manual de Usuario: mensaje de error9, Configurar Herramienta

Importante en los servicios de indexación es el parámetro fuenteConsulta que, dependiendo de la fuente del servicio de indexación, permite procesar la consulta que se vaya a realizar sobre él. Esto es, en el caso de crear traducciones al francés de textos en español, y considerando que se tiene un servicio de indexación cuya fuente es el documento traducido, es decir, la visión y no el documento base, si se desea realizar una consulta sobre el índice creado por el servicio en español, va a ser necesario realizar el mismo proceso con la consulta que con el documento inicial. Se tendrá que traducir la consulta del español al francés.

Se deben cubrir todos los campos del formulario, de no ser así el error que mostrará será:

Figura 212: Manual de Usuario: mensaje de error10, Configurar Herramienta

3.4.3.3 Paso 3: Indicar los parámetros de los servicios

En este paso se deben cumplimentar los parámetros propios de cada uno de los servicios que se proporcionan. Hay que indicar que la realización de esa comprobación es responsabilidad del usuario que implementó las clases.

En el momento de pulsar este último botón "*Modificar*", se procede a realizar la modificación del fichero de configuración así como de la colección.

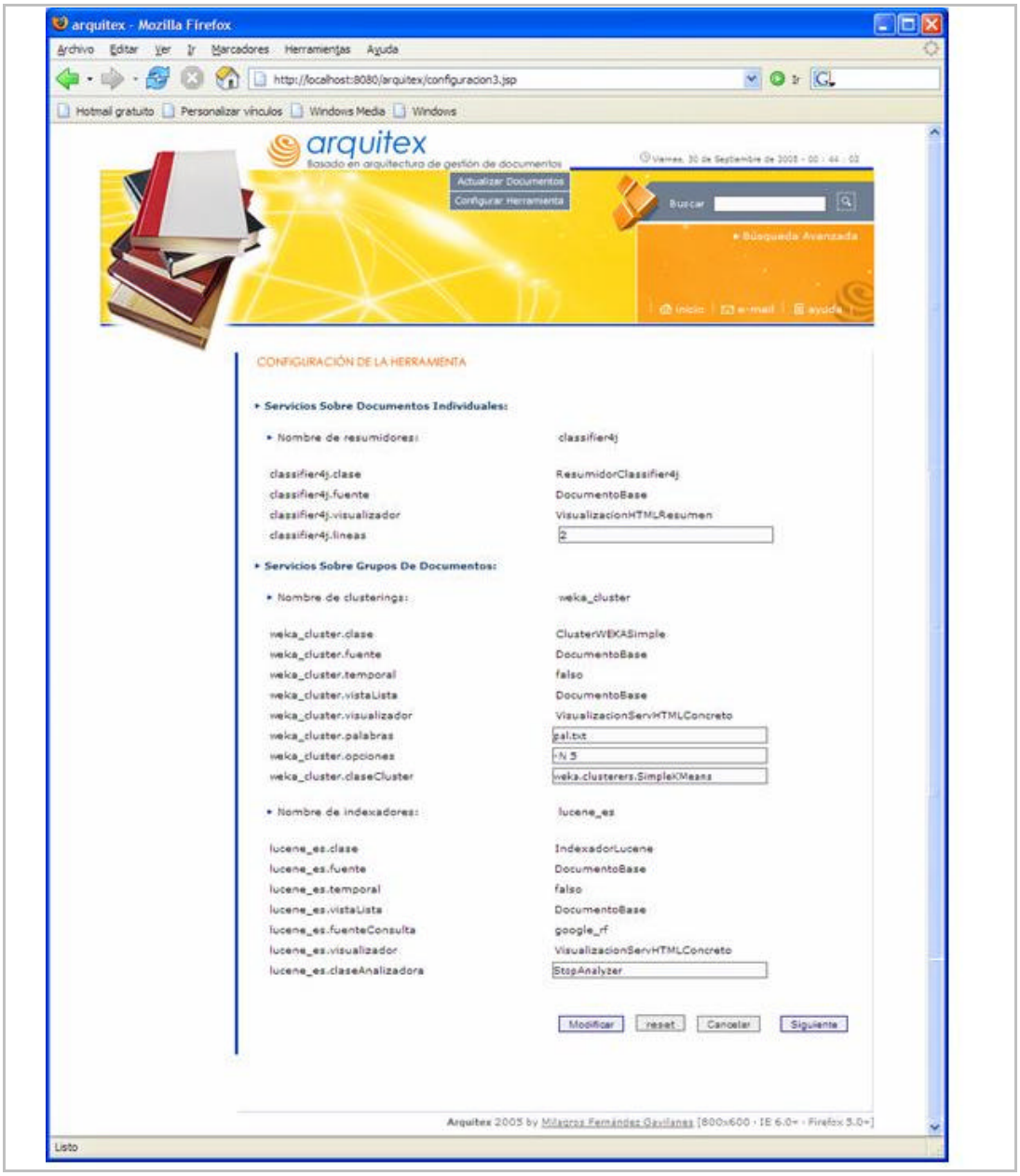


Figura 213: Manual de Usuario: Configurar Herramienta, ventana 3

A este nivel, son los propios desarrolladores de las clases concretas quienes deben de realizar la comprobación. Por ejemplo, en la figura se puede observar en el servicio Lucene que el parámetro propio es una clase Analizadora. Ahora bien, si dicha clase no pertenece a la clase Analyzer, éste dará un error.

Figura 214: Manual de Usuario: mensaje de error11, Configurar Herramienta

3.4.3.4 Paso 4: Indicar los componentes de la cadena, el acceso directo y otros

Este paso es el último de los que se realiza en la configuración de la herramienta.

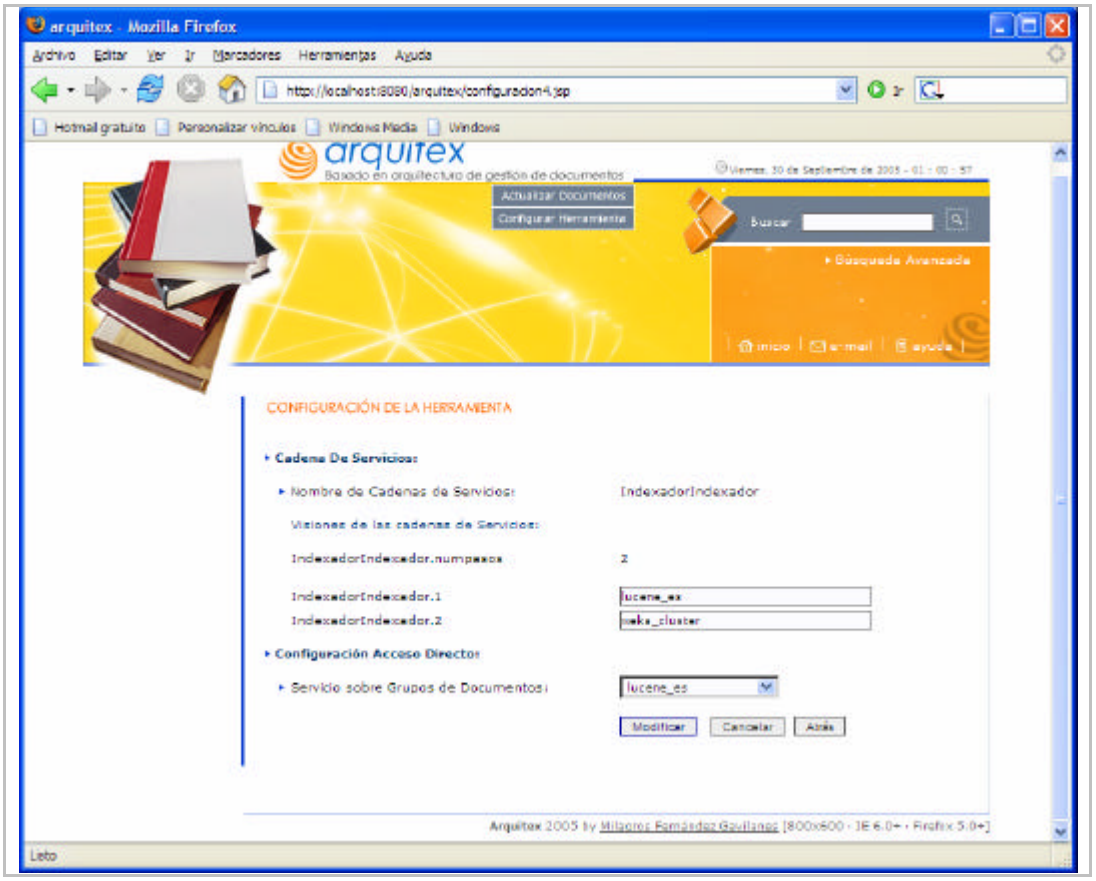


Figura 215: Manual de Usuario: Configurar Herramienta, ventana 4

En el caso de haber indicado que iba a existir cadenas de servicios y de haber pasado el número de servicios que se iban a usar, es necesario cumplimentar los campos indicando el nombre del servicio que utilizar. Por tanto los nombres que se vayan a introducir tienen que forzosamente pertenecer a los servicios de grupos de documentos. En caso contrario, un mensaje de error mostrará:

Error: verifique que los pasos de las cadenas sean servicios de grupo de documentos

Figura 216: Manual de Usuario: mensaje error 12, Configurar Herramienta

Si ha incorporado alguna fuente o una fuenteConsulta de un servicio de indexación en alguno de los servicios de documentos individuales, será necesario definirlo a este nivel:

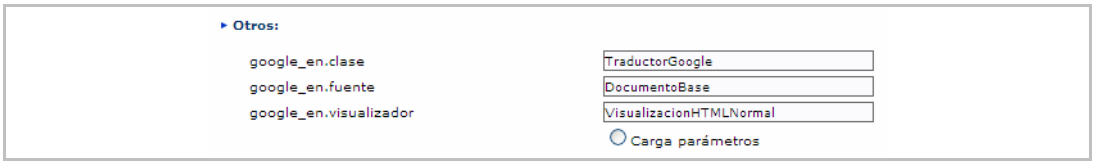


Figura 217: Manual de Usuario: Configurar Herramienta, otros

En el momento de pulsar "*Carga parámetros*", se cargarán los parámetros si ya existen en el fichero de configuración. Si no es así los campos estarán vacíos dando lugar a:

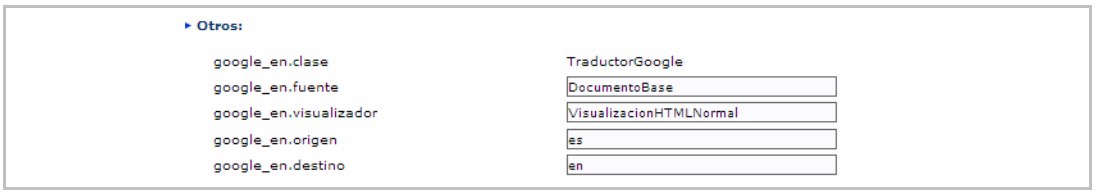


Figura 218: Manual de Usuario: Configurar Herramienta, otros2

Finalmente es necesario configurar el acceso directo de la herramienta, para ello se utiliza una lista de desplegables. Solo hay que seleccionar uno. En caso de no poner nada, se tomará por defecto la seleccionada.

3.4.4 Búsqueda Avanzada

Para realizar una búsqueda, o bien se selecciona el acceso directo que previamente se habrá configurado, o bien se pulsa el botón "*Búsqueda Avanzada*", en cuyo caso se muestra la ventana:

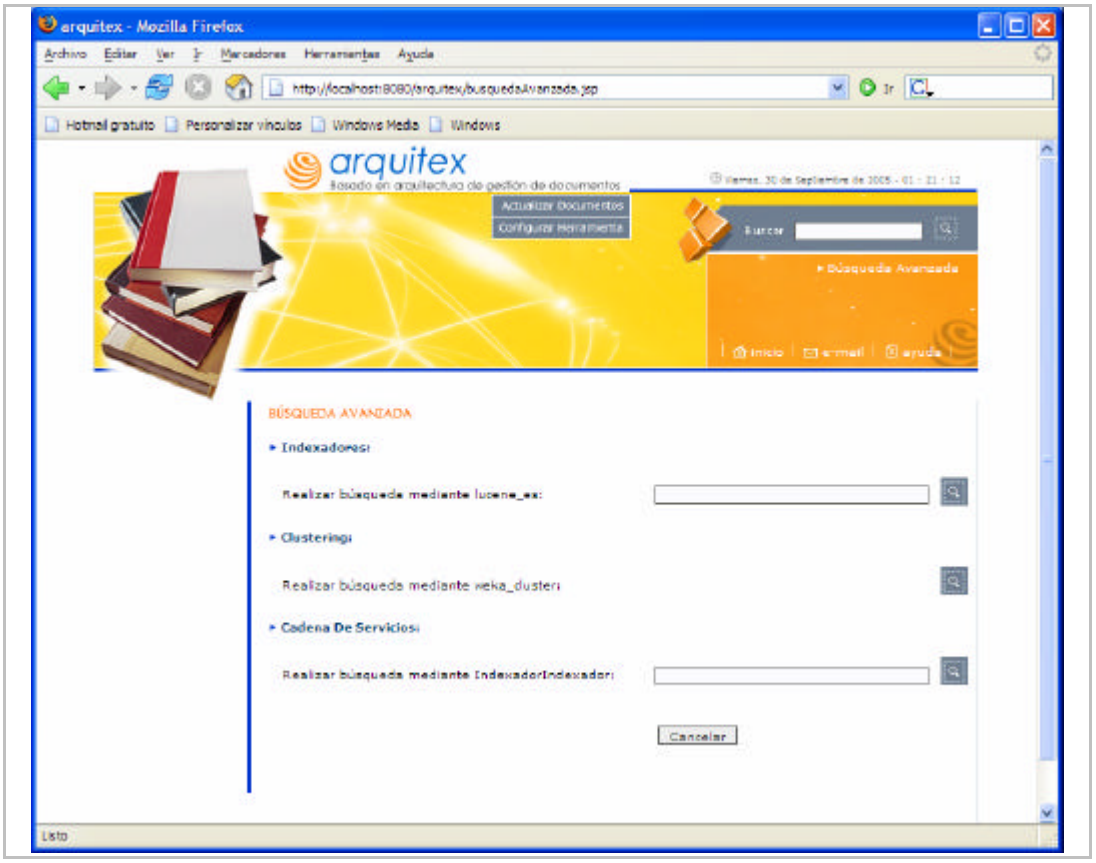


Figura 219: Manual de Usuario: Búsqueda Avanzada

Como se observa en la figura, existen en el ejemplo un indexador, un cluster, y una cadena de servicios. Si se decide realizar una consulta mediante el indexador, dependiendo de la fuenteConsulta, se procesará la consulta para posteriormente mostrar el resultado.

Si se decidiese consultar el clustering en cambio, no se procesaría ninguna consulta ya que éste no lo necesita. Si se realiza una cadena de servicios, y ésta contiene como paso realizar una indexación, también sería necesario procesar la consulta.

Lo que está claro es que cuando se trate de una indexación o de una cadena, va a ser necesario insertar dicha consulta, por lo que esta no puede ser vacía. En caso contrario se muestra:

Error: el campo no puede estar vacío. Inténtelo de nuevo

Figura 220: Manual de Usuario: mensaje error 13, Búsqueda Avanzada

3.4.5 Resultados de una búsqueda

Se va a distinguir los tres casos:

- Si desea visualizar los resultados de un indexador, lo que se va a mostrar es:

Figura 221: Manual de Usuario: resultado búsqueda indexador

- En el caso de que se desee visualizar los resultados de un clustering, lo que se va a observar es:

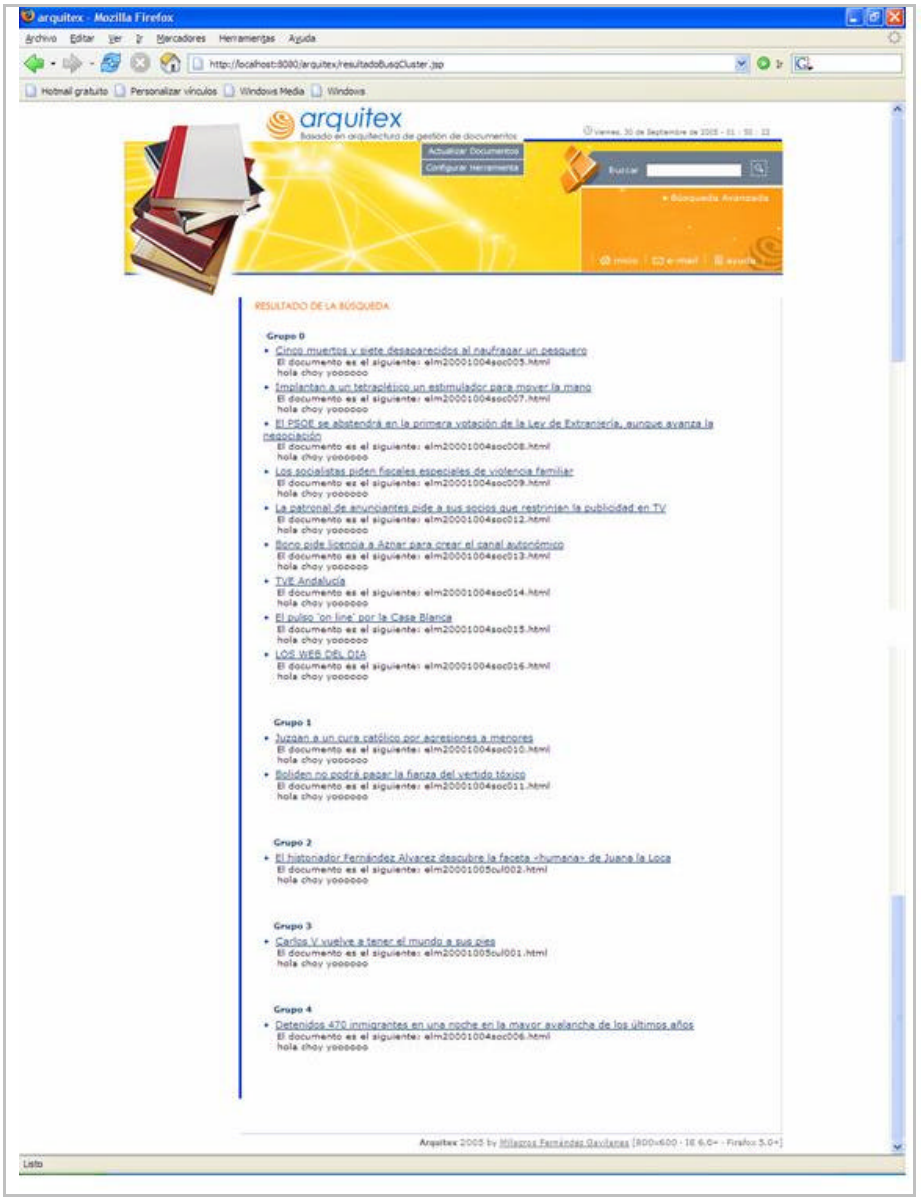


Figura 222: Manual de Usuario: resultado búsqueda clustering

En el caso del clustering, devolverá tantos grupos como se le indique en el fichero de configuración.

- Si desea visualizar los resultados de una cadena, dependiendo de cual sea el último paso, se visualizará igual que alguno de los dos anteriores.

Si no existe ningún documento asociado a la consulta que se está realizando, se mostrará:

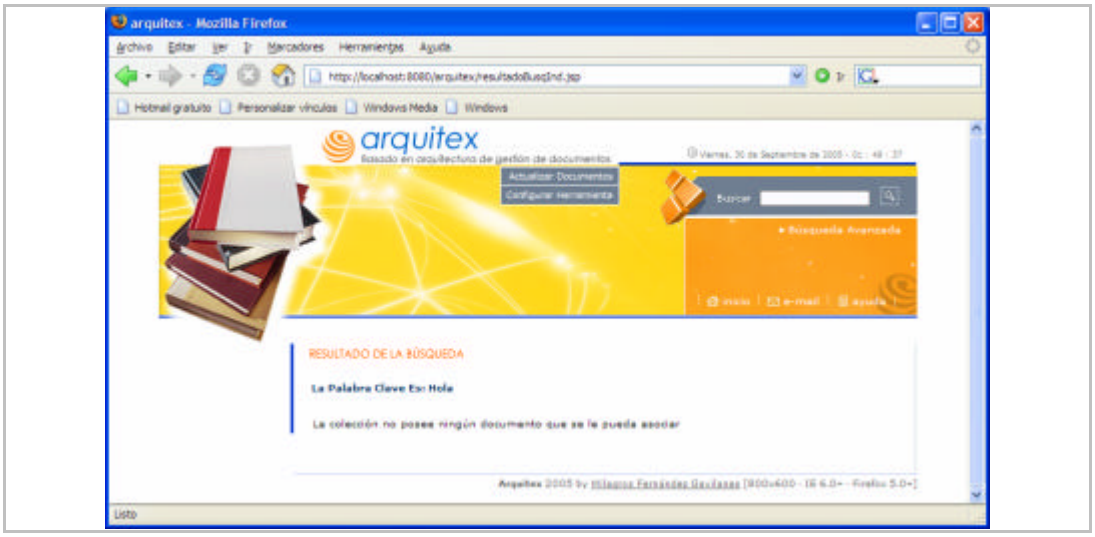


Figura 223: Manual de Usuario: mensaje error 14, Resultado Búsqueda

3.4.6 Visualización de los documentos

Una vez realizada la búsqueda sobre un servicio, y mostrado el resultado, solo nos queda seleccionar el documento que se quiere visualizar. Como ejemplo tenemos:

Figura 224: Manual de Usuario: visualización de los documentos

3.4.7 Configuración incorrecta o fichero inexistentes

En caso de arrancar la aplicación y de que el fichero de configuración no exista o está defectuoso, aparecerá una ventana informando del mismo.

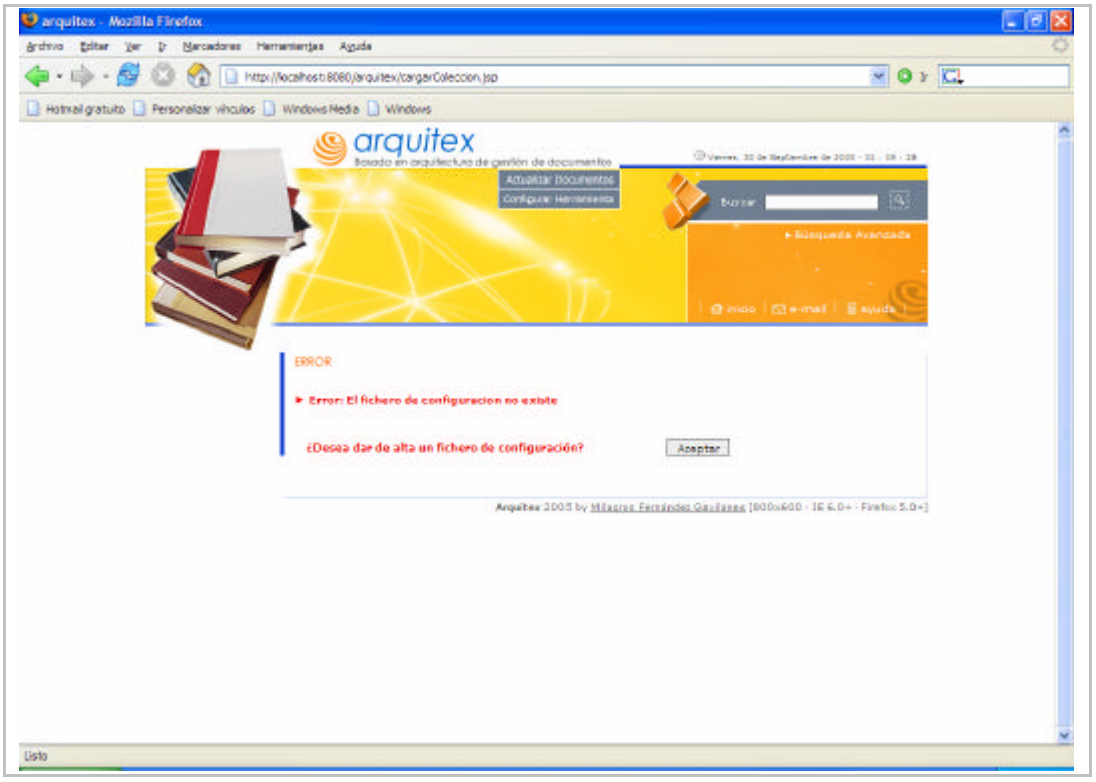


Figura 225: Manual de Usuario: Error de fichero de configuración

En el momento de *aceptar*, se iniciaría una página tal y como se ve en el apartado de configuración con la diferencia de que no existe ningún parámetro cubierto, pero además, en el último paso, se crea el fichero de configuración.

Lo mismo ocurre cuando el fichero de configuración se encuentra defectuoso.