Universidad de Valladolid

E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática de Sistemas

Metabuscador sobre fuentes de datos bajo un esquema HTML

Autor:
Daniel Ballesteros González

Universidad de Valladolid

E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática de Sistemas

Metabuscador sobre fuentes de datos bajo un esquema HTML

Autor:

Daniel Ballesteros González

Tutora:

Mercedes Martínez González

Resumen

Con el aumento de los sistemas de información es necesario nuevos métodos de acceso a los datos cuando esta no se brinda directamente para su manipulación. A las nuevas formas de acceder a la información mediante servicios web, hay que añadirle la manipulación de páginas web mediante mecanismos automáticos que agilicen y simplifiquen al usuario final la obtención de esa información. En este trabajo se ha procedido a crear un sistema completo que recopila información de múltiples fuentes de datos con una interfaz web. Para ello, se han estudiado las fuentes de datos y se han implementado una serie de wrappers para comunicarse y extraer información de ellas. Por último, para facilitar el acceso a esos datos por parte del usuario cuando han sido recopilados se ha llevado a cabo la creación de una base de datos relacional que permita el almacenamiento de esta información. Como mecanismo de interacción con el usuario final se ha creado una aplicación web que enlaza con el sistema de extracción de datos y sirve los datos al usuario.

Índice

Índice de figuras

1. Introducción

1.1. Motivación

Con el incremento en los últimos años de páginas web que muestran información al usuario cada vez se ha visto más la necesidad de la creación de herramientas que aunen toda esa información para ponerla a disposición del usuario en un menor tiempo y con un menor coste físico. La finalidad de la obtención de dicha información puede ser mucho más que mostrarla al usuario entrando en este aspecto temas como el big data. Esta información se puede obtener de diferentes fuentes siendo la más común la disposición de la información que se encuentra en el propio sistema o el acceso mediante servicios web. El ámbito de actuación de esta aplicación pasa por la obtención de dicha información a través de diferentes interfaces web.

A la hora de crear aplicaciones para la obtención de grandes cantidades de información de una fuente de datos que no es propia, hay que tener en cuenta varios factores. Por un lado lo común en estos casos es no disponer ni del esquema de los datos ni del formato en el que se encuentran. Por otro lado el tiempo de acceso a esas fuentes de datos es clave ya que a medida que pasa el tiempo es más probable que el usuario abandone la aplicación o le resulte menos atractiva. Uno de los retos que han motivado la decisión de crear un sistema que realice estas tareas es la de poder superar los dos puntos anteriores.

Para llegar a conocer el esquema de las fuentes de datos y el formato de los datos que obtengo se va a proceder al estudio pormenorizado de cada fuente de datos. Por último para conseguir resultados en el menor tiempo posible se va a optimizar el proceso de obtención de páginas web de las fuentes de datos.

1.2. Objetivos del sistema

Los objetivos generales de un metabuscador son los de reunir información de diferentes fuentes de datos para integrarla bajo un único esquema de destino. Para llevar a cabo esta tarde estos sistemas se dedican a inspeccionar diferentes fuentes de datos para despues extraer dicha información. A mayores de la funcionalidad implicita en un sistema de integración de datos, como la interfaz de este sistema va a ser una interfaz web uno de los objetivos va a ser dotar al sistema de un mínimo de requisitos del que disponen la mayoría de sitios web. Entre esa funcionalidad de la que tiene que disponer el sistema debido a su tipo de interfaz está la gestión de usuarios. Otro de los aspectos importantes dada la naturaleza de esta aplicación, es la de mostrar los productos por lo que se va a utilizar la aplicación web para este propósito.

Por último, se tiene que diseñar la aplicación teniendo en cuenta que todo lo anterior se realice tanto en el menor tiempo posible como con un nivel de fallos mínimo. El lograr que una aplicación como esta que depende de fuentes de terceros se ejecute rápidamente es un reto muy difícil de alcanzar, esto es debido a que no se puede garantizar la fiabilidad de unas fuentes de terceros y también entra en juego las características del sistema.

2. Estado del arte

En los últimos años se ha visto que internet se ha convertido en la mayor base de datos de información siendo incrementada diariamente. A pesar de que la gente cada vez tenga más acceso a internet y más dominio sobre el uso de los ordenadores sigue habiendo un problema que no se soluciona con la formación del usuario, el usuario cada vez demanda más información en menos tiempo. El problema que deriva de lo anterior es que el acceso a la información almacenada en los sistemas actuales está empezando a ser suplantada por el acceso mediante herramientas que permitan multiplicar la información obtenida reduciendo el tiempo.

Cuando una persona busca información sobre un viaje o sobre un vuelo no suelen acceder a la fuente de datos originales. Para recuperar información que les sea útil hacen uso de cualquier herramienta (comunmente web) que les permita recuperar información de cientos o miles de fuentes de datos simultaneamente para evitar la pesada tarea de ir una a una. En este punto es donde entran en juego los metabuscadores, estos productos se elaboran con la finalidad de hacer la vida más fácil al usuario ya sea ahorrandoles tiempo como ahorrándoles dinero.

Gracias a estos sistemas los usuarios acceden mediante una interfaz común a una serie de fuentes de datos abstrayendoles de problemas relativos a localización, idioma, disponibilidad, etc. Los metabuscadores pueden obtener la información de diferentes formas:

- Llegando a acuerdos con las empresas y obteniendo los datos directamente de la fuente
- Mediante servicios web que ponen a disposición los dueños de los datos
- Creando sistemas que hagan uso mediante técnicas de programación de los datos a través de las interfaces web que proporcionen

De los casos citados en el punto anterior el más fácil es el primero ya que no solo se dispone de los datos que se quieren obtener, se dispone también de los esquemas de los datos y su formato. Otro punto a favor de este sistema es que la empresa que lo realiza se asegura que el acceso a los datos es fiable en todo momento, cuenta con la aprobación de la empresa que suministra los datos y no se van a intentar mecanismos por parte de esta para "cortar" el suministro de datos.

La segunda opción está reservada a aquellas empresas que no tengan ningún inconveniente en "abrir" las puertas de sus almacenes de información (o al menos de una parte). Estas empresas suelen estar interesadas en que la gente conozca sus datos ya sea para que repercuta en el volumen de ventas o por cualquier otro motivo. Desgraciadamente esta opción no es la más común a pesar de que diariamente hay empresas que adopten este modelo.

Por último la opción más fácil de poder llevar a cabo es la tercera. Es la más fácil de llevar a cabo porque no se necesita un acuerdo con la empresa ni que esa empresa haya adoptado un modelo "abierto" con respecto a sus datos. El único requisito necesario es que la empresa de la que se quiere obtener la información tenga una interfaz web, algo que es bastante común. Si la empresa dispone de ese requisito es posible obtener esos datos mediante la creación de alguna herramienta

que automatice esa tarea. Uno de los problemas de este método es que es bastante dependiente de la política de la empresa ya que en cualquier momento podrían cambiar sus condiciones de uso, cambiar la interfaz web (o quitarla directamente) o cambiar algo que desencadene en que deje de funcionar esa herramienta.

Una vez que se ha visto las formas de obtención de información de las fuentes de datos se plantean dos políticas para obtener esa información, integración virtual y data warehousing [2]. En el data warehousing los datos de las fuentes individuales son cargados en una base de datos y es sobre esos datos sobre los que se ejecutan las queries para obtenerlos. En la integración virtual en cambio los datos permanecen en las fuentes y se accede a ellos cuando se les necesita.

La integración virtual tiene sus propios componentes que difieren de los del data warehouse en muchos puntos. Los componentes que se ocupan de comunicarse con la fuentes de datos se conocen como wrappers y su tarea es la de enviar las consultas a la fuente de datos, recibir las repuestas y aplicar alguna modificación (si es necesaria) a la respuesta. En esta arquitectura el usuario interactua con el sistema de integración a través del esquema intermedio. El esquema intermedio está creado específicamente para la aplicación y contiene aquellos aspectos del dominio que son relevantes para la aplicación. Con este esquema intermedio se consigue mostrar una visión unificada de los datos de las diferentes fuentes a pesar de que el esquema de esas fuentes pueda variar entre ellos pasando de XML a bases de datos relacionales.

En el data warehousing a los componentes que se ocupan de comunicarse con las fuentes de datos se les conoce como herramientas ETL. Estas herramientas se encargan de extraer, transformar y cargar (extract, transform and load) la información de las fuentes en el warehouse. Estas herramientas se comportan como una tubería, como un procesamiento en cadena de los datos. Son más complejas que un wrapper ya que no solo se tienen que comunicar con las fuentes de datos sino que probablemente tengan que realizar transformaciones a esos datos que puede implicar limpieza, transformación de valores, etc. Al almacen que contiene la información y sobre el que el usuario lanza las consultas se le conoce como data warehouse.

Cuando se habla de data warehousing, como he comentado antes, las herramientas encargadas de "tratar" con las fuentes de datos se conocen como ETL. Estas herramientas se encargan de refrescar la información de la fuente de datos a través del tiempo. Algunas de sus tareas son el aplicar filtros a los datos que se obtienen, transformar datos entre diferentes formatos, deducir cuando una misma tupla se refiere a la misma entidad, etc.

Como se ha comentado en los puntos anteriores, tanto en una arquitectura como en otra, hay componentes que necesitan comunicarse con las fuentes de datos. En el caso de que la fuente de datos fuera propia se haría uso de la herramienta pertinente (un driver JDBC en caso de ser una base de datos MySQL por ejemplo), o crear un cliente que consumiera un servicio web. Como en esta aplicación la comunicación va a ser contra páginas web se necesitan herramientas que permitan comunicarse con las páginas web, a estas herramientas se las conoce como webdrivers. La funcionalidad de estas herramientas es la de simular el comportamiento de un navegador

web en un programa. Gracias a los webdrivers es posible utilizar la potencia de un lenguaje de programación unido a la manipulación de páginas webs.

Para obtener la información relevante de una página web hay varias opciones, desde el tratamiento de cadenas puro hasta técnicas más avanzas como son lenguajes de consulta sobre documentos HTML como pueda ser XPath o CSSQuery. En el tratamiento de cadenas lo más común es utilizar expresiones regulares para obtener la información relevante. El problema de este sistema es que es poco flexible a los cambios en la interfaz y es difícil de mantener debido al uso de expresiones regulares. Como se ha comentado antes, un sistema más evolucionado es el de hacer consultas sobre la página que se obtenga con lenguajes diseñados para este propósito.

3. Análisis

3.1. Requisitos

3.1.1. Requisitos funcionales

- El sistema deberá permitir la creación de usuarios.
- El sistema deberá permitir eliminar usuarios.
- El sistema deberá permitir la modificación de la información de los usuarios.
- El sistema deberá permitir la búsqueda de productos a usuarios registrados.
- El sistema deberá permitir que el usuario salga de la aplicación en cualquier momento de la navegación.
- El sistema deberá permitir que el usuario acceda a su cuenta mediante un formulario de loggin.
- El sistema deberá permitir la búsqueda de información en varias páginas de forma transparente para el usuario.
- El sistema deberá mantener un registro de las búsquedas de cada usuario a lo largo del tiempo.
- El sistema deberá mantener un registro de los productos encontrados a lo largo del tiempo.
- El sistema deberá mantener un registro del histórico de precios de cada producto almacenado.
- El sistema deberá permitir eliminar toda la información que el sistema mantiene de un usuario.
- El sistema deberá permitir el almacenamiento de la misma búsqueda para un usuario en diferentes momentos de tiempo.

3.1.2. Requisitos no funcionales

- El sistema deberá poder ejecutarse bajo navegadores que ejecuten javascript.
- Robustez: El sistema deberá responder adecuadamente ante datos incorrectos introducidos por el usuario.
- Seguridad: El sistema deberá mantener la seguridad de las contraseñas.
- Instalación: Junto a la herramienta se entregará el manual de instalación.
- Respuesta a fallos: El sistema deberá mostrar un aviso al usuario cuando se produzca un fallo interno.

3.2. Casos de uso

3.2.1. Actores

- Usuario registrado: Persona registrada en el sistema y que quiere utilizar el sistema con todas las posibilidades que ofrece.
- Usuario no registrado: Persona que no se ha registrado en el sistema, lo único que puede hacer este actor es registrarse en el sistema.
- Sistema: Encargado de llevar a cabo las labores de mantenimiento del sistema.

3.2.2. Diagrama de casos de uso

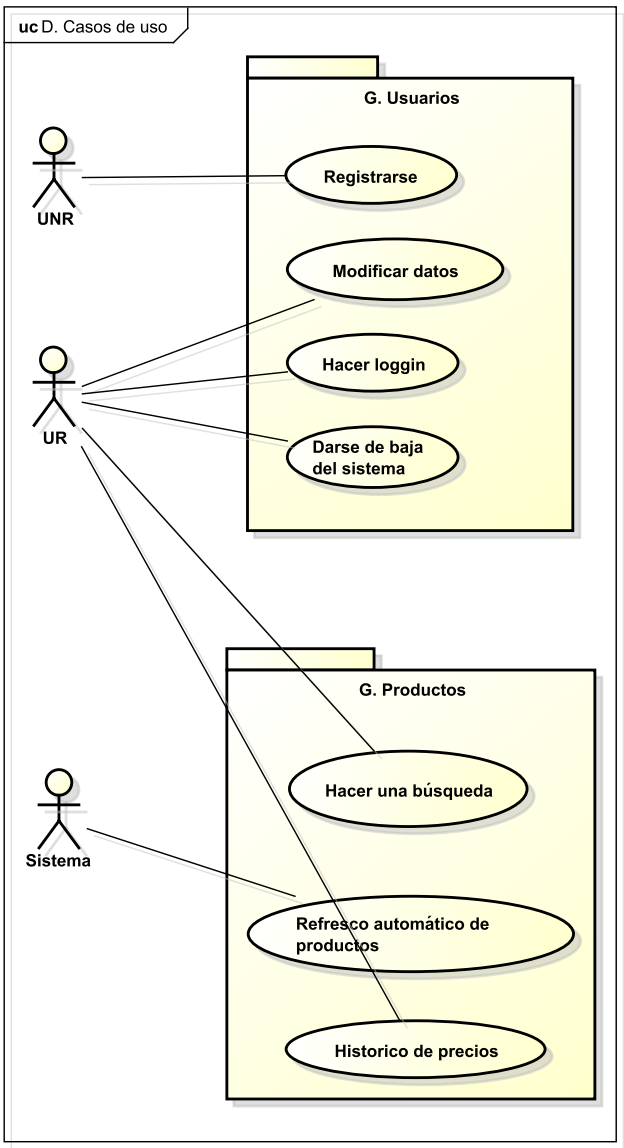

Figura 1: Diagrama de casos de uso

3.2.3. Especificación de casos de uso

Caso de uso: Registrarse

Actores implicados

- Usuario no registrado

Precondiciones

No hay precondiciones asociadas a este caso de uso.

Flujo normal

1. El usuario solicita una petición de registro en el sistema.
2. El sistema le proporciona un formulario estandar con los datos que necesita de él.
3. El usuario rellena dicho formulario y lo envía de vuelta al sistema.
4. El sistema comprueba que los datos ingresados tengan el formato correcto.
5. El sistema inserta los datos del usuario en el sistema.
6. El sistema redirige al usuario a la pantalla de login.

Postcondiciones

- El usuario queda registrado en el sistema.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir. El sistema se lo indicará al usuario llevandole a una pantalla de error.
- **Caso alternativo 2**
  - En el paso #4 el sistema comprueba que alguno de los campos no se ajustan al formato que el sistema espera.
  - El sistema le comunica al usuario la incidencia y el motivo del error
- **Caso alternativo 3**
  - En el paso #5 de este caso de uso el sistema comprueba que el mail que ha introducido el usuario ya está en uso y se lo comunica al usuario.

Caso de uso: Modificar datos

Actores implicados

- Usuario registrado

Precondiciones

El usuario debe encontrarse en su página principal dentro de la aplicación.

Flujo normal

1. El usuario dentro de su página principal solicita modificar sus datos.
2. El usuario modifica los datos que quiera y envía los datos al sistema.
3. El sistema comprueba el formato de los datos.
4. El sistema registra los datos y envia al usuario a la página principal de la aplicación.

Postcondiciones

- Los nuevos datos del usuario quedan registrados en el sistema.
- El usuario se encuentra en su página principal.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir. El sistema se lo indicará al usuario llevandole a una pantalla de error.
- **Caso alternativo 2**
  - En el paso #4 de este caso de uso el sistema comprueba que los datos introducidos por el usuario no son válidos y se lo comunica al usuario.
- **Caso alternativo 3**
  - En el paso #4 de este caso de uso el sistema comprueba que el mail que ha introducido el usuario ya está en uso y se lo comunica al usuario.

Caso de uso: Hacer login

Actores implicados

- Usuario registrado

Precondiciones

El usuario debe estar registrado en el sistema.

Flujo normal

1. El usuario solicita una petición de login en el sistema.
2. El sistema presenta al usuario un formulario para que ingreses sus datos de accesso.
3. El usuario ingresa sus datos y los envía al sistema.
4. El sistema comprueba si los datos son válidos.
5. El sistema lleva al usuario a su pantalla principal.

Postcondiciones

- El usuario está en su página principal.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir. El sistema se lo indicará al usuario llevandole a una pantalla de error.
- **Caso alternativo 2**
  - En el paso #4 de este caso de uso el sistema comprueba que los datos introducidos por el usuario no son válidos y se lo comunica al usuario.
  - El sistema informa al usuario de que no se puede acceder a la aplicación con tales datos y le lleva a la página principal de la aplicación.

Caso de uso: Darse de baja del sistema

Actores implicados

- Usuario registrado

Precondiciones

El usuario debe encontrarse en su página principal dentro de la aplicación.

Flujo normal

1. El usuario solicita al sistema que le dé de baja.
2. El sistema elimina toda la información que contenga del usuario.

Postcondiciones

- Toda la información del usuario ha sido eliminada del sistema.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir. El sistema se lo indicará al usuario llevandole a una pantalla de error.

Caso de uso: Hacer una búsqueda

Actores implicados

- Usuario registrado

Precondiciones

El usuario debe estar registrado en el sistema.

Flujo normal

1. El usuario ingresa los datos del producto sobre el que quiera consultar.
2. El sistema realiza una búsqueda del producto que ha introducido el usuario.
3. El sistema muestra los resultados al usuario.

Postcondiciones

No hay postcondiciones asociadas a este caso de uso.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir. El sistema se lo indicará al usuario llevandole a una pantalla de error.
- **Caso alternativo 2**
  - En el punto #1 del caso de uso el sistema detecta que la cadena de búsqueda no cumple con los requisitos mínimos.

Refresco automático de productos

Actores implicados

- Sistema

Precondiciones

No hay precondiciones asociadas a este caso de uso.

Flujo normal

1. El sistema obtiene las búsquedas que se han llevado a cabo en el sistema.
2. El sistema ejecuta todo el proceso implicado en la obtención de los datos de las distintas fuentes.
3. El sistema guarda los nuevos precios de los productos asociados a esa búsqueda.

Postcondiciones

No hay postcondiciones asociadas a este caso de uso.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir, el error quedar registrado en los logs del sistema.

Histórico de precios

Actores implicados

- Usuario registrado

Precondiciones

El usuario tiene que haber hecho una búsqueda que genere resultados.

Flujo normal

1. El usuario solicita consultar el historial de precios de ese producto.
2. El sistema consulta los últimos siete precios del producto y su precio máximo y mínimo.
3. El sistema consulta la información del producto.
4. El sistema muestra al usuario los datos obtenidos.

Postcondiciones

No hay postcondiciones asociadas a este caso de uso.

Flujos alternativos

- **Caso alternativo 1**
  - En cualquier paso del caso de uso el sistema puede experimentar un error interno que le impida seguir, el error quedar registrado en los logs del sistema.

4. Diseño

4.1. Diseño de la arquitectura

4.1.1. Descomposición en subsistemas

El patrón arquitectónico que se ha elegido para el sistema es el patrón de capas. Con este patrón cada capa utiliza las funcionalidades de las capas inferiores, es posible sustituir cualquiera de las capas por implementaciones alternativas, se minimiza la dependencia entre capas al crear interfaces en cada capa que disminuyen el acoplamiento entre capas, etc.

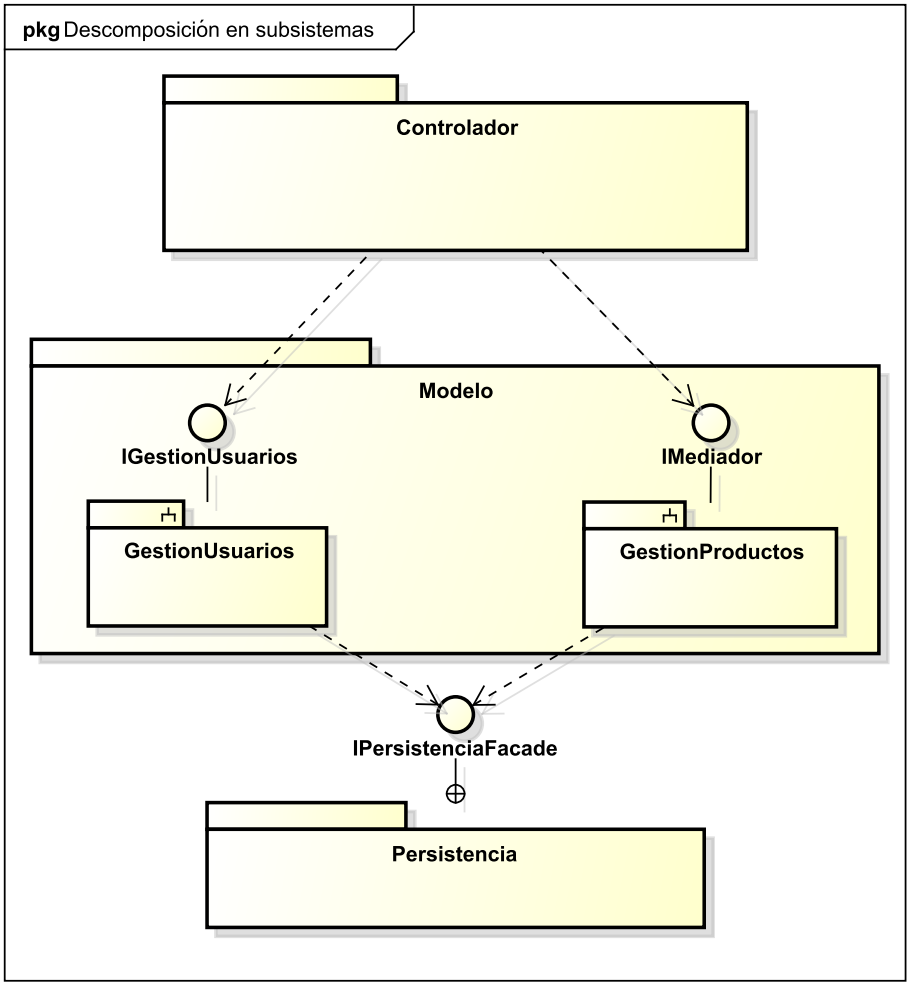

Figura 2: Descomposición en subsistemas

En el diagrama se pueden ver las 3 capas de las que consta la aplicación:

- Controlador: Contiene las clases que reciben las peticiones de los usuarios y se encargan de delegar la funcionalidad en las capas inferiores.

- Modelo: En esta capa se encuentran las clases del modelo de dominio de la aplicación, dentro de esta capa hay dos paquetes fundamentales que son el encargado de gestionar las acciones que se llevan a cabo con los usuarios ("GestionUsuarios") y las tareas que se llevan a cabo con los productos ("GestionProductos").

- Persistencia: Esta capa es la encargada de comunicarse con el mecanismo de persistencia que se utiliza en la aplicación. En el caso concreto de esta aplicación es el intermediario entre las clases del modelo y el sistema gestor de base de datos.

4.1.2. Diagrama de despliegue

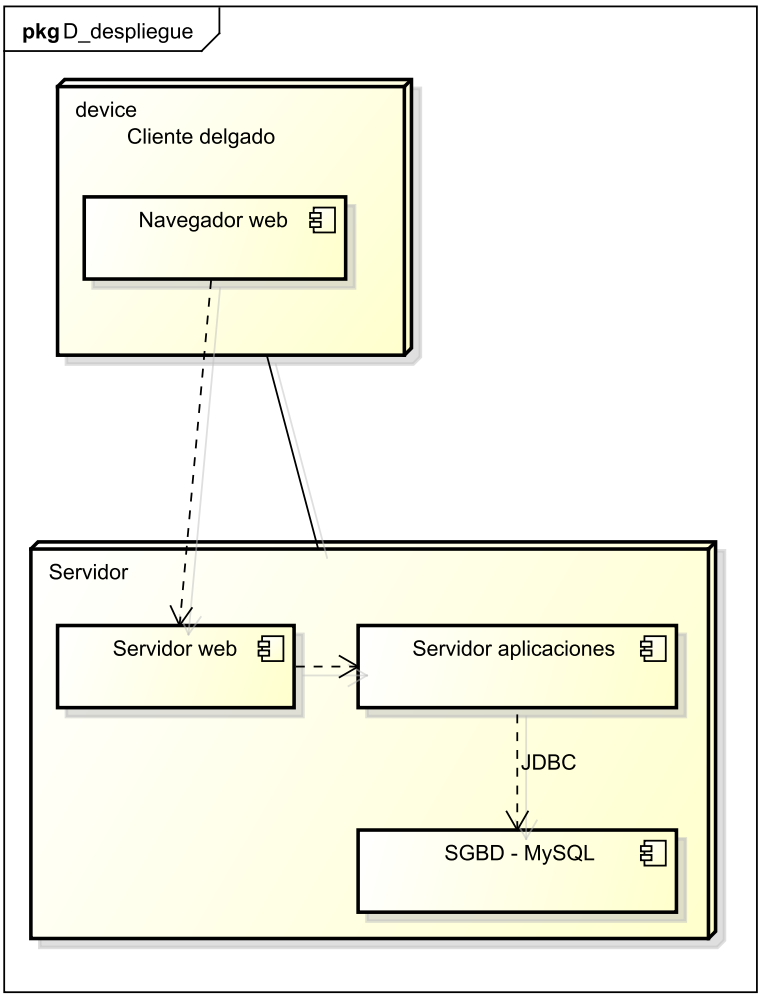

Figura 3: Diagrama de despligue

En el diagrama de despligue se puede ver como están repartidos los componentes de forma física. El navegador hace las peticiones al servidor web que a su vez utiliza el servidor de aplicaciones donde está alojada la aplicación. El servidor de aplicaciones utiliza el sistema gestor de base de datos MySQL como mecanismo de persistencia.

4.1.3. Diagrama de clases

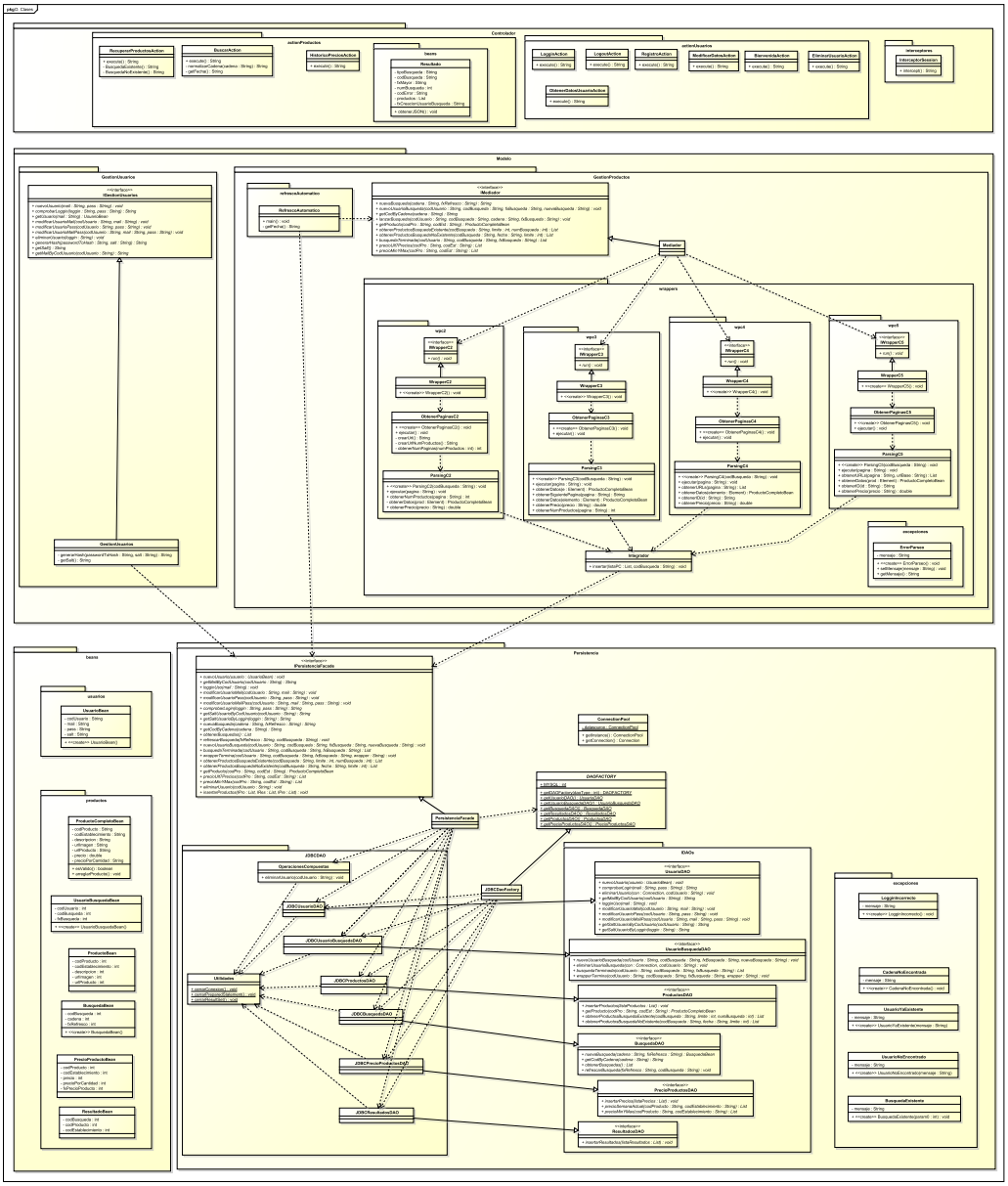

Figura 4: Diagrama de clases

4.1.4. Diagramas de secuencia

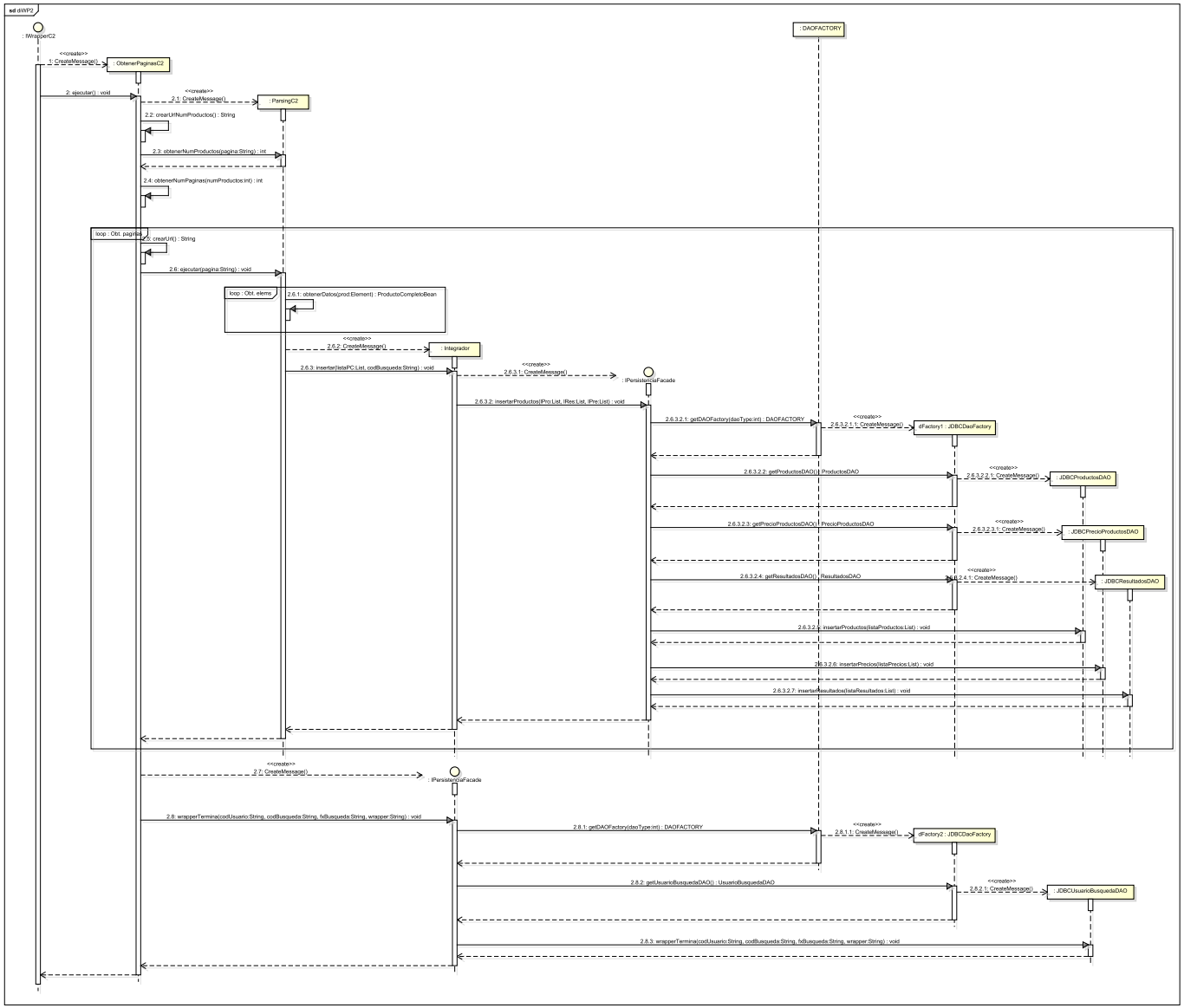

Figura 5: Interaction use: Wrapper 2

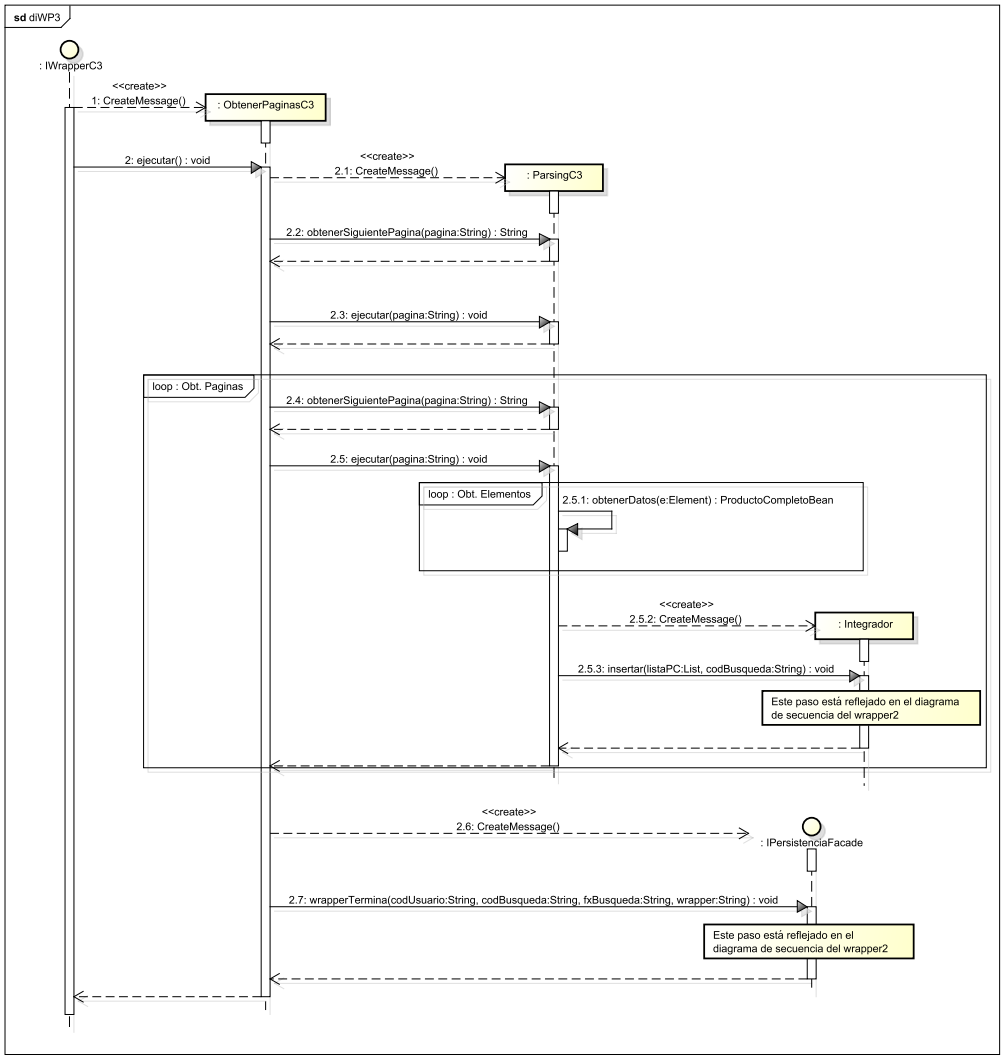

Figura 6: Interaction use: Wrapper 3

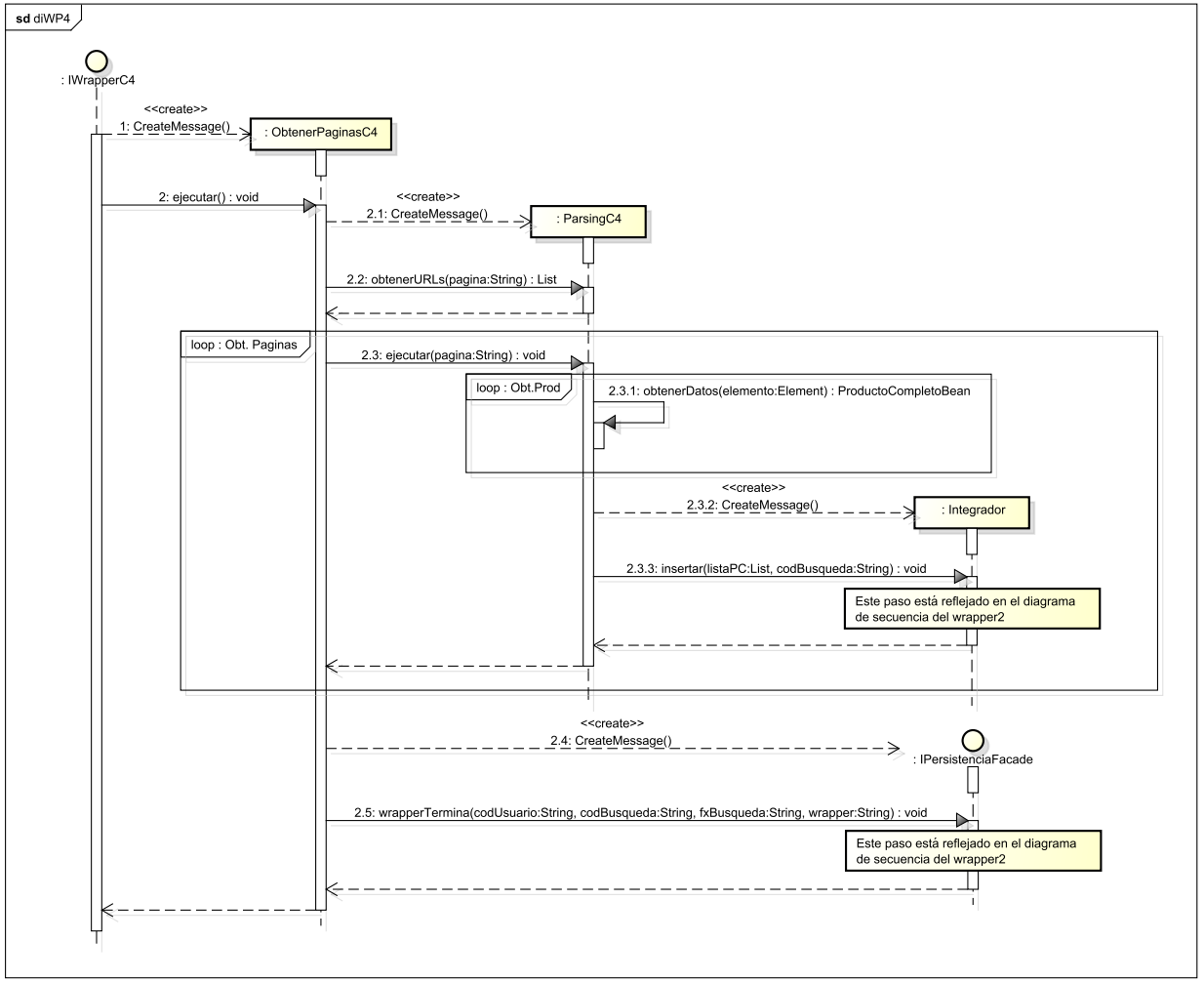

Figura 7: Interaction use: Wrapper 4

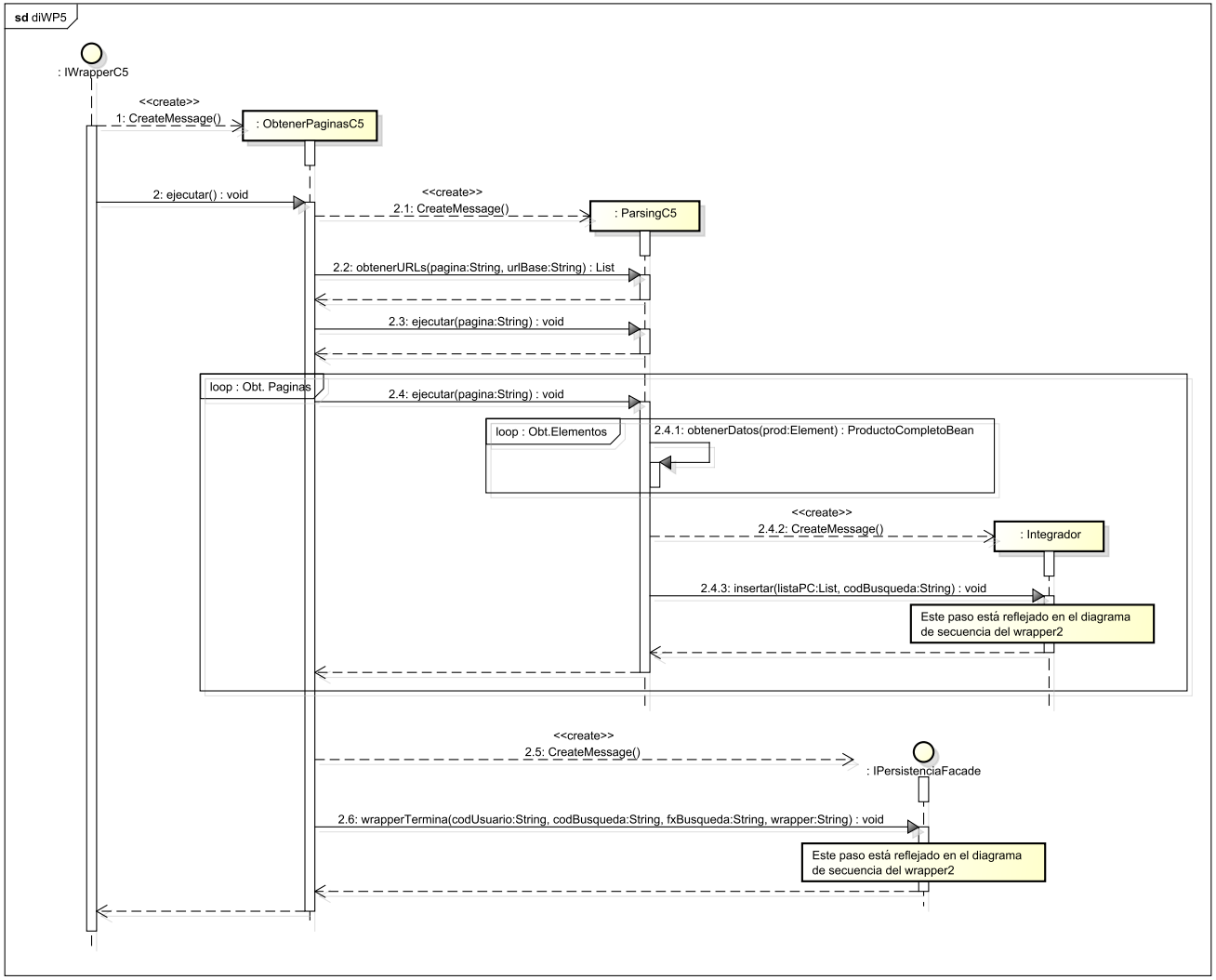

Figura 8: Interaction use: Wrapper 5

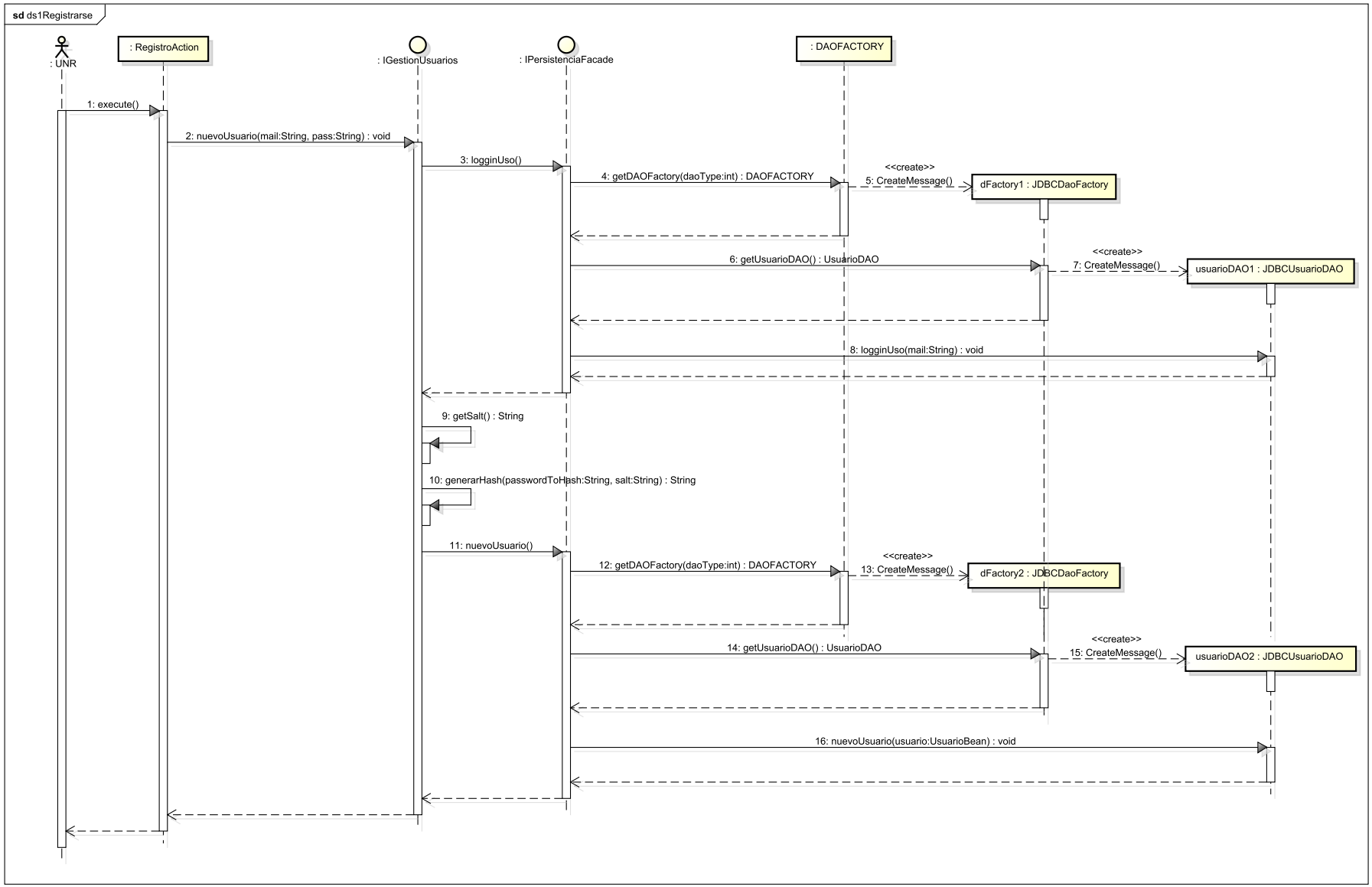

Figura 9: Diagrama de secuencia: Registro de un usuario

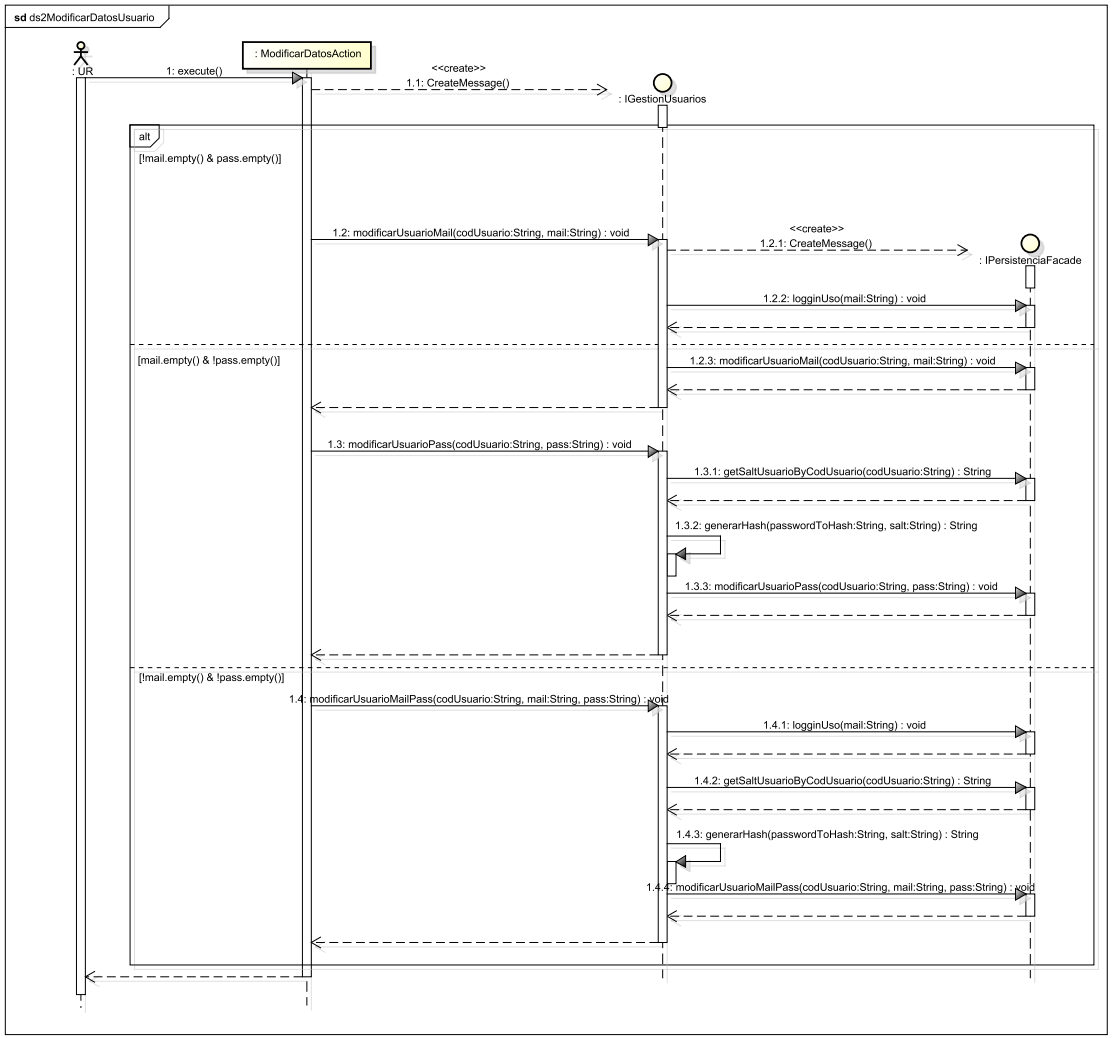

Figura 10: Diagrama de secuencia: Modificar datos del usuario

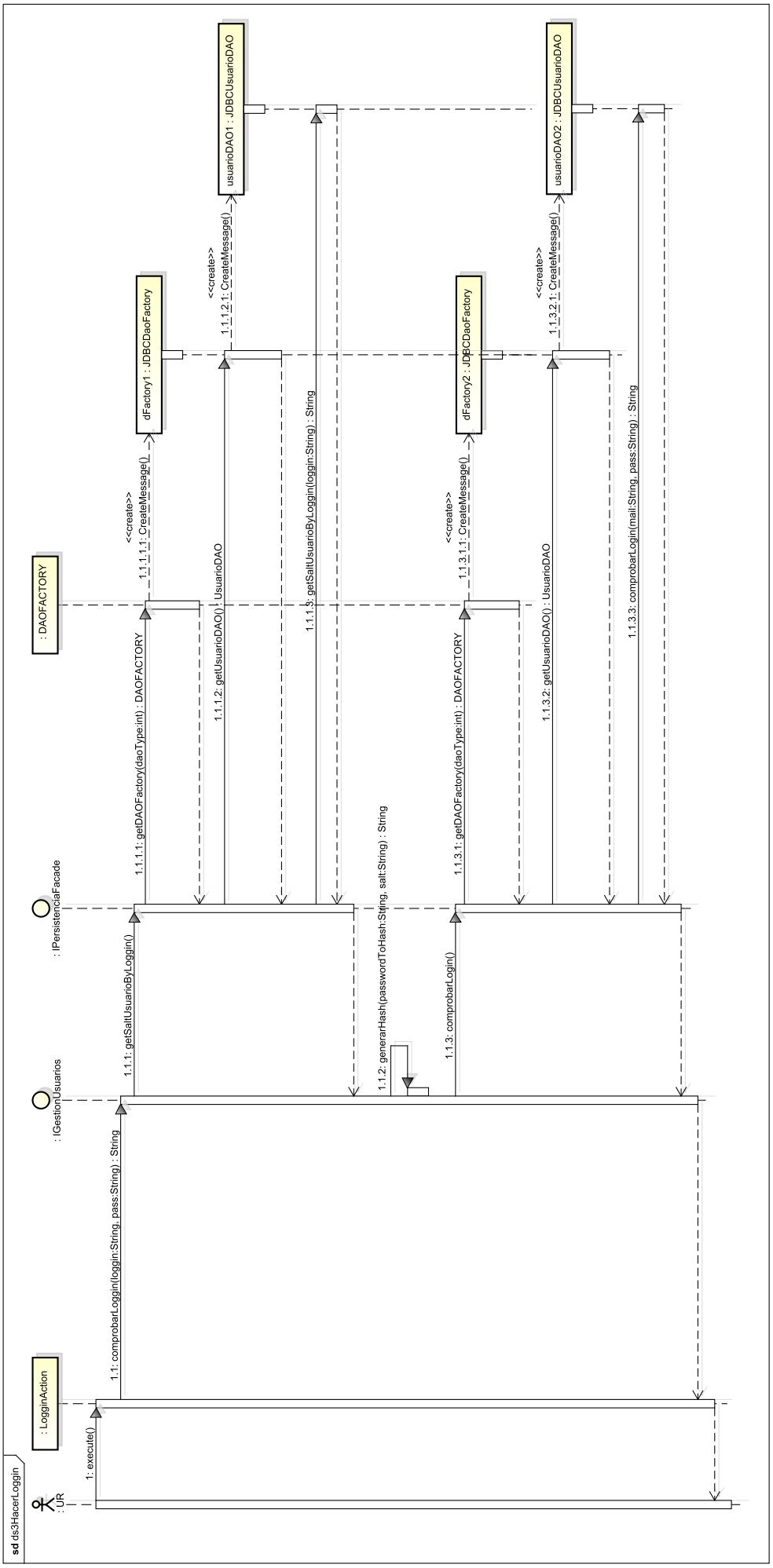

Figura 11: Diagrama de secuencia: Loggin de un usuario

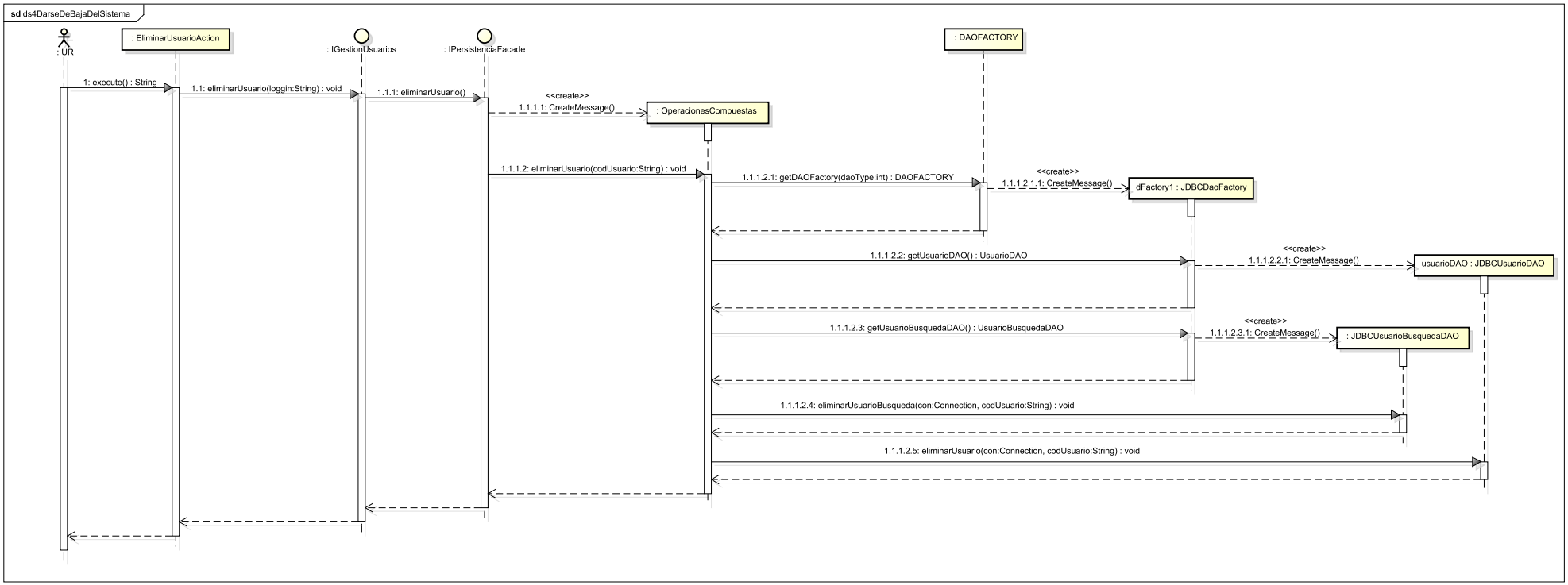

Figura 12: Diagrama de secuencia: Darse de baja del sistema

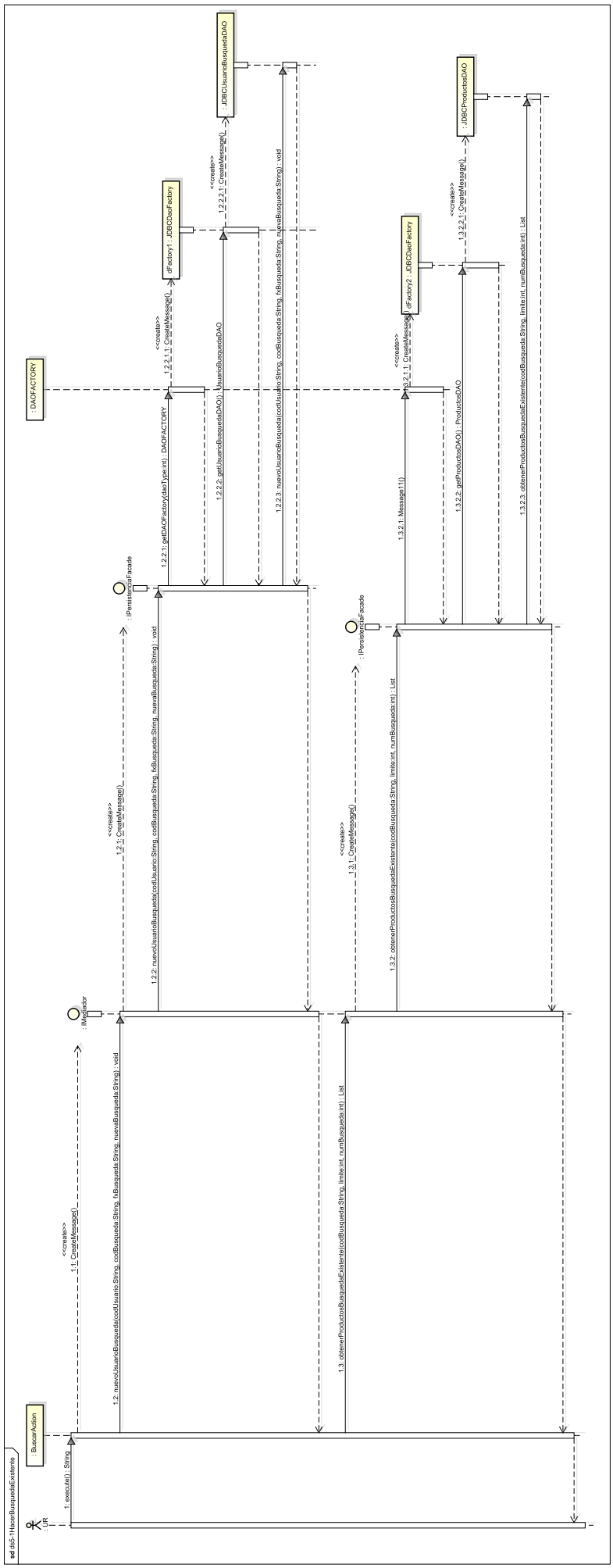

Figura 13: Diagrama de secuencia: Hacer una búsqueda ya existente

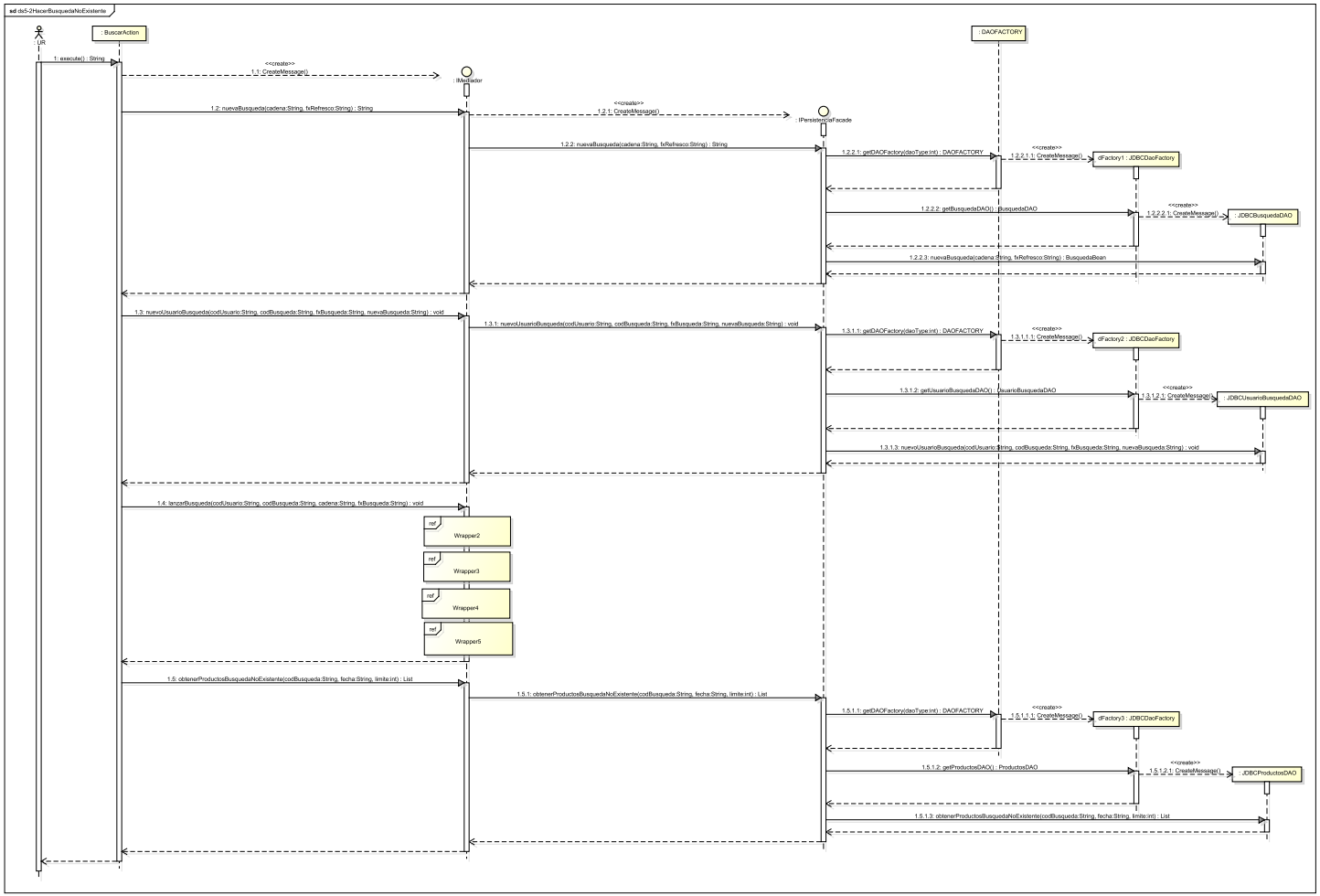

Figura 14: Diagrama de secuencia: Hacer una búsqueda no existente

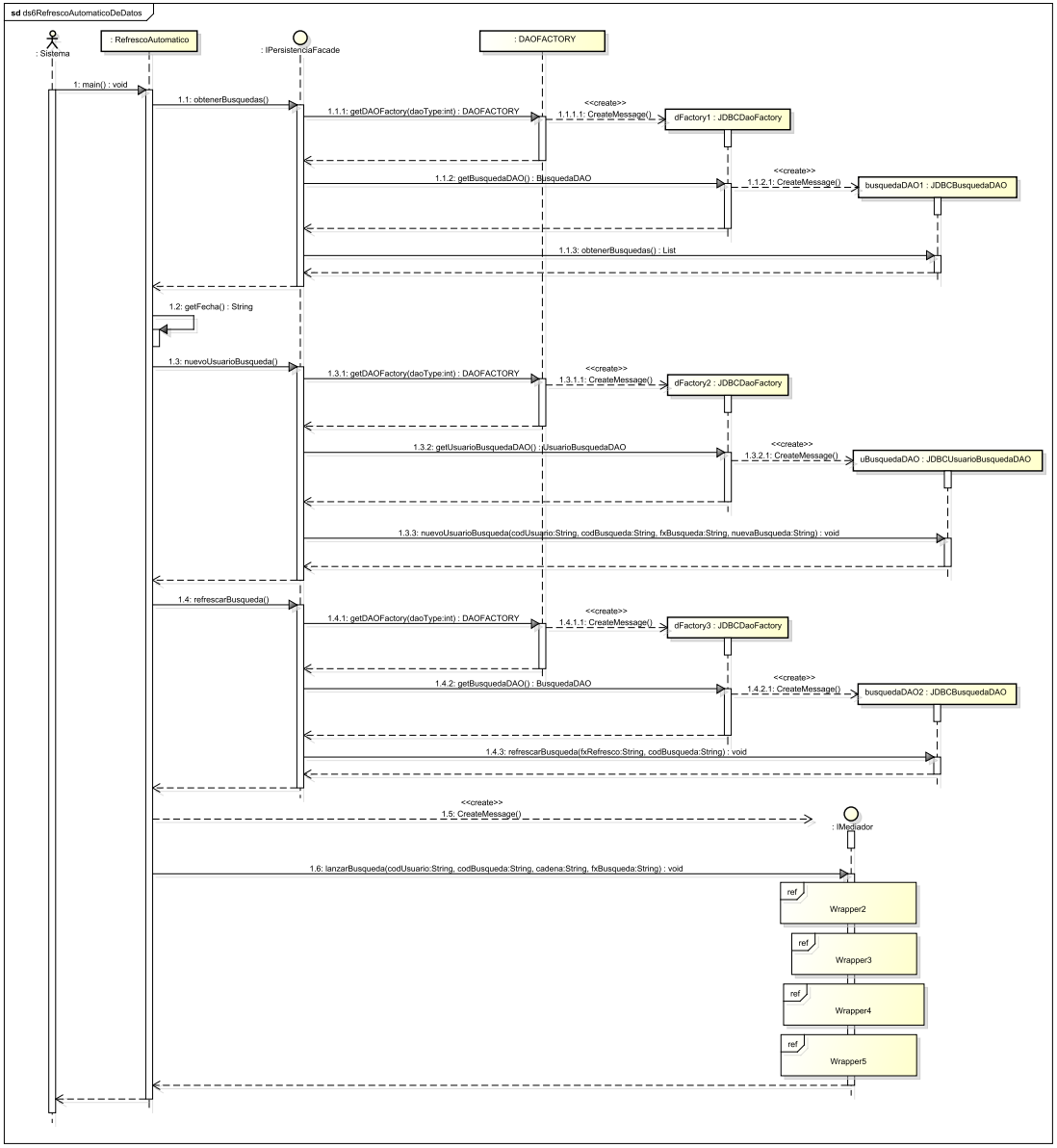

Figura 15: Diagrama de secuencia: Refresco automático de datos

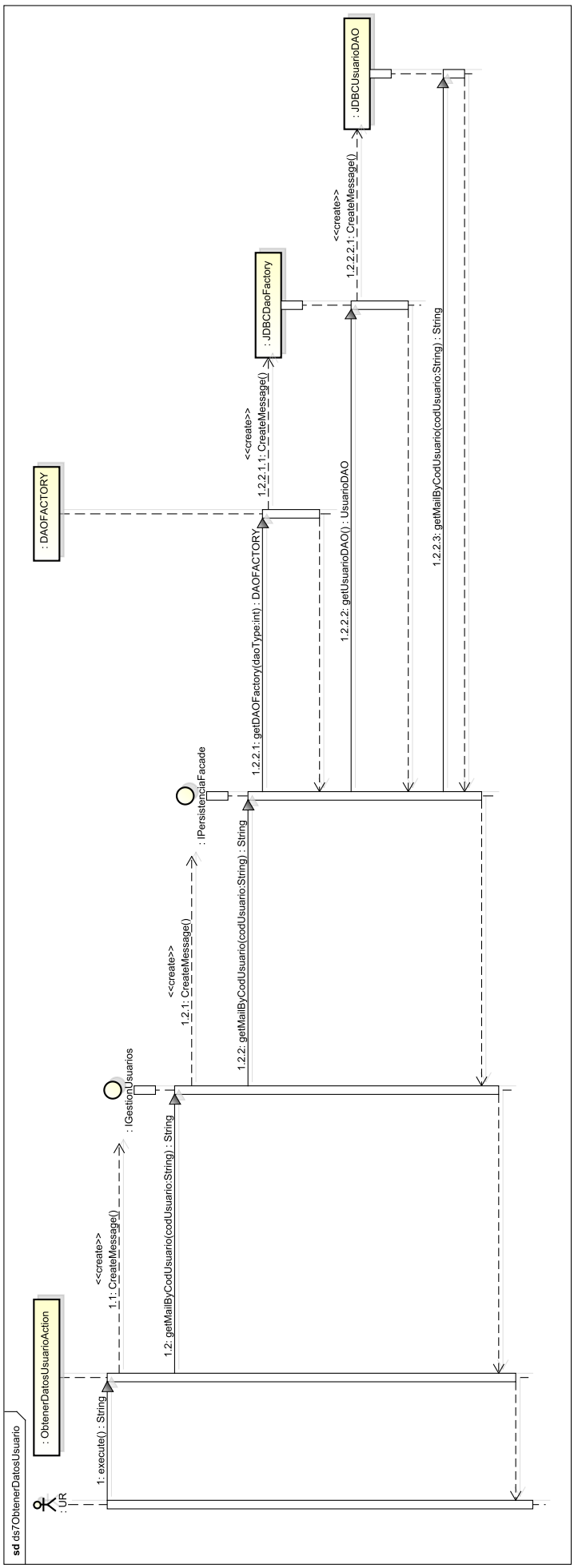

Figura 16: Diagrama de secuencia: Obtención de datos del usuario

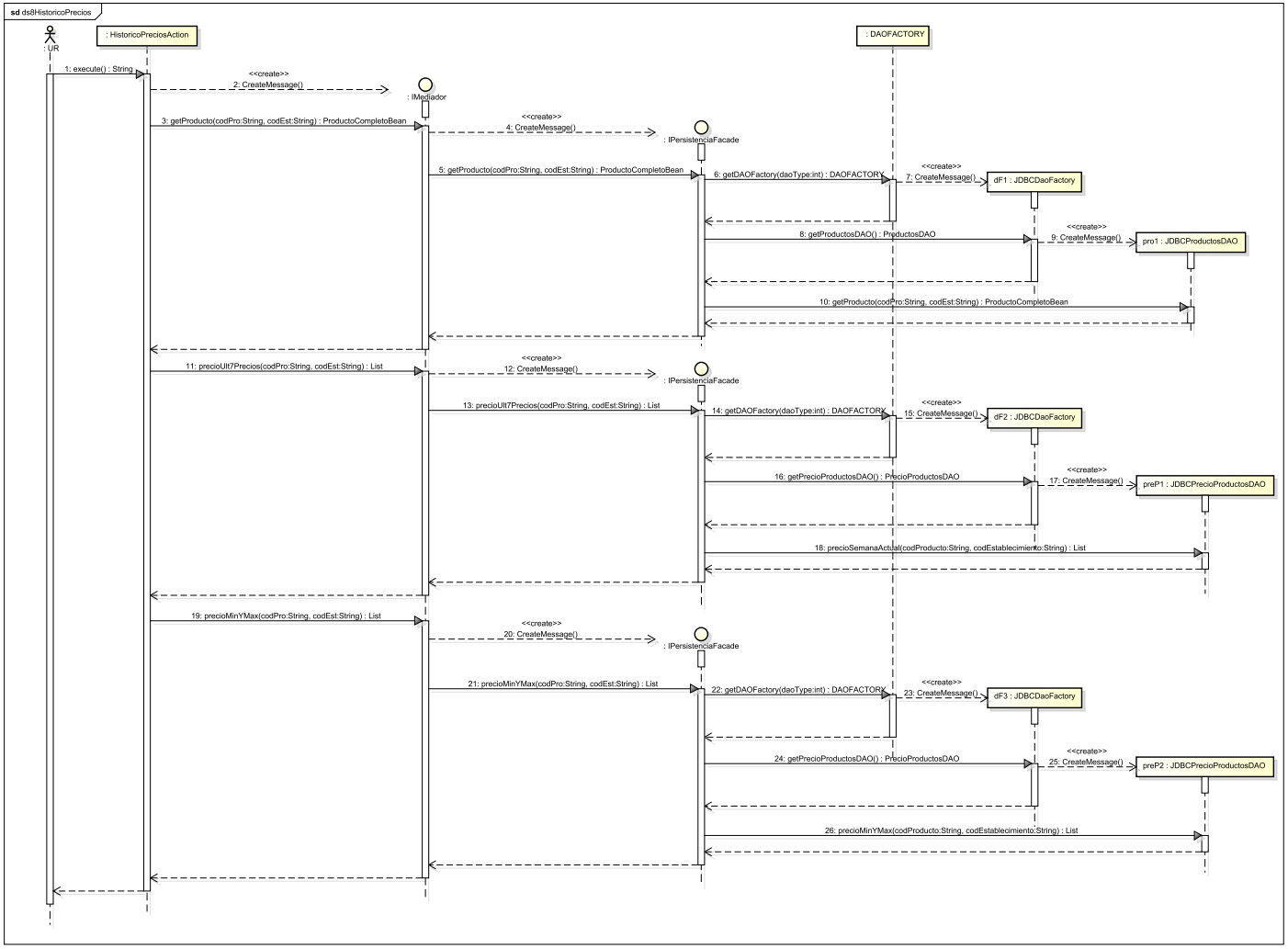

Figura 17: Diagrama de secuencia: Histórico de precios de un producto

4.2. Diseño de la solución

Arquitecturas en la integración de datos

Data Warehouse Con esta arquitectura los datos de las fuentes son cargados en una base de datos física sobre la que se ejecutan las consultas y que es periódicamente refrescada. Uno de los problemas derivados de este sistema es que hay que tener mucho cuidado con el periodo de refresco de los datos y esto depende de la naturaleza de la aplicación.

Integración virtual En este caso los datos son obtenidos de las fuente de datos bajo demanda. Un problema bastante visible de este sistema es que se pueden crear bastantes duplicidades a la hora de proporcionar los datos. Este sistema es adecuado cuando el tiempo de acceso a la fuente de datos es muy bajo. No es conveniente usar esta arquitectura en casos como el de esta aplicación porque el acceso a los datos tarda varios segundos y puede no dar la mejor sensación al usuario.

Decisión A la hora de tomar una decisión final sobre la arquitectura de la aplicación en lo relativo a la integración de la información me decanté por el data warehousing. Este sistema me permite mantener un registro de los distintos precios de los productos y el tiempo de acceso a las fuentes de datos es demasiado elevado como para ir a ellas a por la información cada vez que se necesite.

Solución en la aplicación

Búsqueda de productos

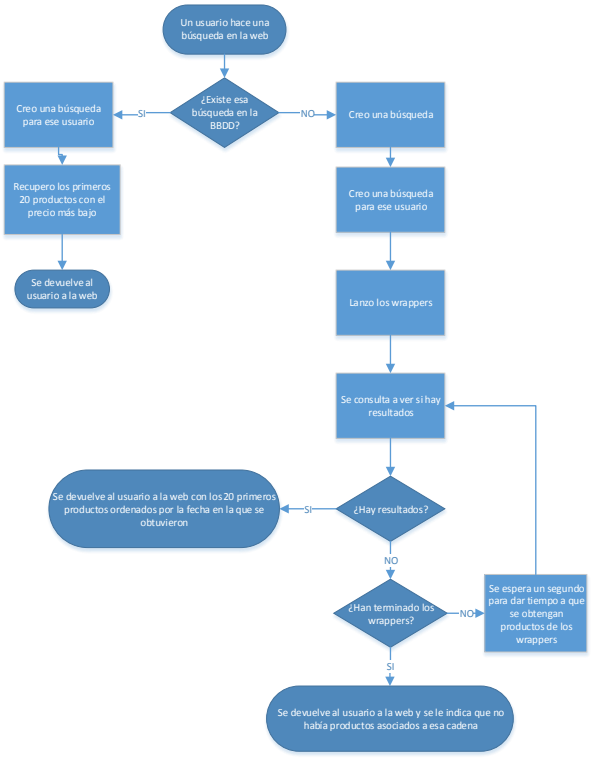

Figura 18: Diagrama de flujo de la búsqueda de productos

Como se puede ver se trata diferente las búsquedas que ya existen en base de datos de las que no. Cuando se realiza una búsqueda se comprueba si esa búsqueda ya se ha hecho, si ya se ha hecho se utilizan los resultados que están disponibles en la base de datos en vez de ir a buscarlos a las fuentes de datos ya que el proceso es mucho más lento. Si están disponibles en el sistema esos datos se sirven directamente al usuario. Por otro lado si nunca se ha hecho esa búsqueda hay que ir a las fuentes de datos para obtener los resultados, insertarlos en el warehouse y servirlos al usuario. El usuario entonces tendrá que esperar hasta que haya resultados disponibles. Una vez que se ha hecho esa primera búsqueda el sistema se va a encargar de refrescar los resultados de las búsquedas que se hayan hecho para que en posteriores búsquedas se acceda a la base de datos en vez de a las fuentes de datos.

Recuperación de productos

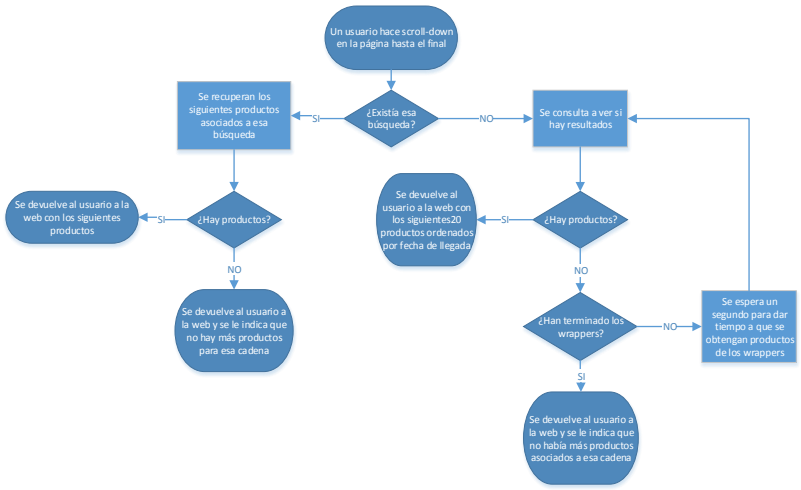

Figura 19: Diagrama de flujo de la recuperación de productos

Como se puede ver en los diagramas una de las principales diferencias entre las dos posibilidades que se dan al hacer una consulta es cómo se devuelven los resultados al usuario. Si la búsqueda ya existía en base de datos se devuelven al usuario los resultados ordenados por precio ascendente mientras que si para obtener los datos se está yendo a las fuente de datos se devuelven los datos en orden de llegada. Esto se hace así porque uno de los objetivos en mente a la hora de realizar esta solución era la de mostrar contenidos al usuario lo antes posible y si se está yendo a las fuentes de datos y queremos mostrar contenido cuanto antes no podemos garantizar que los primeros resultados que obtengamos sean los que tienen el precio más bajo y podrían obtenerse resultados después con un precio aun menor dando una sensación de confusión al usuario. Este problema no ocurre en el caso en que se utilizan datos que están en base de datos ya que disponemos de todos los datos nada más realizarse la consulta y podemos devolverlos en el orden que queramos.

Cuando se ha hecho la búsqueda y se han devuelto los resultados la búsqueda queda reflejada en la base de datos. El siguiente paso es diseñar un mecanismo por el que se refresquen los datos para esa búsqueda ya que los precios de los productos almacenados pueden variar a lo largo del tiempo, y hay que mantener los datos consistentes con respecto a las fuente de datos originales.

Para ello se lanza periódicamente una fase que vuelve a realizar las búsquedas ya hechas sobre las fuentes de datos. Esta fase se lanza desde un método main y se ejecuta todas las noches. Podría darse el caso de que hubiera búsquedas en la base de datos que no hubieran generado ningún resultado, en ese caso hay dos opciones:

- Realizar esa búsqueda vacía otra vez: Una ventaja de esta opción es que si esa búsqueda se vuelve a repetir por algún usuario va a estar disponible inmediatamente y no hay que ir a las fuentes de datos, por otro lado está suponiendo un coste diario en términos de tiempo y de ciclos de CPU.
- Eliminar esa búsqueda: Eligiendo esta opción penalizamos el tiempo de aquellos usuarios que no sepan que esa búsqueda no genera resultados pero en cambio ahorramos tiempo a la hora de hacer el refresco automático. Llevar a cabo esta opción sería tan fácil como crear un script que se ejecutase antes del refresco automático y que llamara a un procedimiento de la base de datos para que borrara aquellas búsquedas que no han generado resultados. Si se lleva a cabo este sistema estaríamos eliminando parte de la información que tenemos acerca de los usuarios (las búsquedas que llevan a cabo).

En la aplicación he considerado tomar la primera opción ya que considero que si un usuario ha hecho esa búsqueda es porque es de interés para el.

4.3. Diseño de los subsistemas

4.3.1. Diseño del controlador

Definición

Struts2 [6] es un framework MVC para Java. En este framework las tres partes del patrón quedan repartidas de la siguiente forma:

- Modelo: Clases java que representan el modelo de la aplicación, en esta parte se pueden incluir desde EJB hasta JavaBeans e incluso otros frameworks para dar soporte a la aplicación (Hibernate, etc).
- Vista: Representada por JSP.
- Controlador: En este punto es donde entra en juego Struts 2 ya que sustituye los Servlets por clases Java que reciben las peticiones web e interfieren con las clases del modelo.

Actions

Son clases java que hacen la vez de servlet y por tanto reciben las peticiones, tienen que respetar algunas normas:

- Cada atributo debe tener creados sus métodos get y set. Estos métodos son utilizados por el framework para obtener los atributos del action y ponerlos.
- Los métodos que reciban las solicitudes tienen que devolver una cadena de texto para poder decidir cual va a ser la redirección de ese action.
- Normalmente el método que suele recibir la acción se nombra "execute".

En mi aplicación los actions están divididos en paquetes según sirvan para la gestión de usuarios o para la gestión de productos, se ha intentado minimizar la lógica en estas clases intentando delegar la funcionalidad hacia clases de paquetes más bajos.

- actionProductos: Aquí están incluidos los siguientes actions:
  - BuscarAction.java: Esta clase se encarga de realizar una búsqueda sobre las fuentes de datos o sobre la base de datos propia dependiendo de cada caso.
  - HistoricoPreciosAction.java: Esta clase se encarga de obtener los últimos siete precios de un producto y del precio máximo y mínimo de ese producto.
  - RecuperarProductosAction.java: Esta clase se encarga de recuperar los productos de una búsqueda tanto si esta búsqueda existía como si no.
- actionUsuarios:
  - BienvenidaAction.java: Esta clase es la primera que se invoca cuando el usuario entra en la aplicación y sirve para saber si ese usuario ya ha hecho login o no redirigiendolo hacía un `Result` u otro en función de que esté logueado o no.
  - EliminarUsuarioAction.java: Esta clase se utiliza para eliminar toda la información que tenga el sistema de ese usuario, es decir, tanto las búsquedas que ha hecho como la información propia del usuario.
  - LoginAction.java: Esta clase se encarga de comprobar que las credenciales que ha introducido el usuario sean correctas.
  - LogoutAction.java: Esta clase se utiliza para hacer logout en la aplicación.
  - ModificarDatosAction.java: Esta clase recibe las solicitudes relativas a la modificación de datos del usuario.
  - ObtenerDatosUsuarioAction.java: Esta clase es muy sencilla y sirve para obtener el mail del usuario antes de mostrar su perfil.
  - RegistroAction.java: Esta clase es utilizada para registrar al usuario en la aplicación.

Results

Después de que un Action haya sido procesado se debe de enviar una respuesta al usuario, esto se realiza usando results. Un Action puede tener varios results dependiendo de las posibles salidas que tenga el Action que ha finalizado. Gracias a que un Action puede tener varios Results asociados podemos enviar al usuario a una vista distinta dependiendo del resultado de la ejecución del Action o hacer que se descargue un archivo, etc. Un Result puede tener varios tipos depeniendo de a donde queramos mandar al usuario, se le puede mandar a un jsp, a otro Action, etc.

En la aplicación utilizo 3 tipos de result:

- stream: Se utiliza para devolver al usuario un flujo de bytes, en mi caso lo utilizo cuando voy a devolver un json al usuario.
- redirectAction: Se utiliza para redirigir al usuario a otro action, el problema de utilizar este result es que crea una nueva petición para el nuevo Action por lo que se pierden todas las variables que hubiera en el primer action por tanto se puede usar sin problemas cuando no se necesite pasar información de action a otro.
- dispatcher: Se utiliza para enviar al usuario a un JSP.

Interceptores

Son clases que siguen el patrón interceptor y que se encargan de llevar a cabo funcionalidades comunes a varios actions. Como esa funcionalidad es común lo que se hace es sacar la lógica del action y ponerla en una clase que se ejecute antes y después de su ejecución. A pesar de que struts2 cuente con varios interceptores [7] si ninguno de ellos hace lo que necesitamos, podemos crear nuestro propio interceptor y añadirlo al fichero struts.xml para que se ejecute en la invocación de los actions.

En la aplicación he creado un interceptor encargado de comprobar que el usuario esté logueado. Este interceptor se ejecuta antes de cada Action y en caso de que el usuario no esté logueado se le redirige a la pantalla de login y en caso de que si que lo esté se le redirige hacía su destino . Este interceptor se encuentra dentro del paquete controlador.interceptores. Una vez creado este interceptor tengo que añadirle en la pila de interceptores que se ejecuta, este paso lo hago en el fichero struts.xml.

A parte del interceptor que he creado he utilizado dos interceptores encargados de la validación de datos, el interceptor “validation” y el interceptor “workflow”. Estos interceptores se encuentran incluidos en la pila básica de interceptores por lo que no hay que incluirlos. Cuando la validación no es correcta estos interceptores devuelven un result con nombre “input” por lo que por cada Action que use la validación hay que crear un Result para el caso en el que la validación no sea correcta.

Fichero struts.xml

La configuración de Struts 2 se puede hacer mediante un archivo de anotaciones o mediante un archivo XML, lo más normal es utilizar el archivo XML. Este fichero se coloca en el paquete raíz de la aplicación y sirve para mantener una relación entre la URL de una petición, la clase Java que la ejecuta la petición y el result que se ejecutara en cada caso. En este fichero tambien se declaran los interceptores creados por el usuario o se altera el orden de los interceptores ya existentes en caso de que se necesite eso.

Validación de datos

Para la validación de los datos en la aplicación Struts 2 tiene varias opciones:

- Validación mediante un archivo XML

  La validación de los campos en este sistema se realiza con las reglas escritas en un fichero XML en el mismo paquete en el que se encuentra el Action, el nombre debe de seguir la convención "NombreAction-validation.xml". Dentro de este archivo se indica qué campos serán validados y qué validaciones se aplicaran. Algunas de las validacione más importantes que se pueden hacer son:

  - required: Comprueba que el campo no sea nulo.
  - requiredstring: Comprueba que el campo no sea nulo y que la longitud de ese campo sea mayor que cero.
  - stringlength: Comprueba la longitud de una cadena.
  - int, long, short y double: Comprueba que una variable esté dentro de un rango.
  - date: Verifica que la fecha esté dentro de un rango.
  - email: Comprueba el formato del mail.
  - url: Verifica que la url cumpla con el formato.
  - expression: Realiza una validación basada en una expresión.

  Aparte de estas validaciones se pueden hacer validaciones basadas en expresiones, se conocen como validaciones planas, por ejemplo en la aplicación utilizo estas validaciones para comprobar cuando el usuario se registra en la aplicación que las dos contraseñas que ha introducido sean iguales. El validador que se utiliza en este caso es "expression" explicado anteriormente.

- Validación mediante anotaciones: Estas validaciones se indican en el propio Action y se pueden realizar las mismas validaciones que en el caso del XML. Se suelen escribir en el método set del atributo que se quiera validar. Se pueden utilizar tantas validaciones como se necesite al igual que en el caso del XML.

- Validaciones manuales: Para la validación manual Struts 2 proporciona una interfaz llamada "Validateable" que a su vez proporciona un método llamado "validate". Esta interfaz ya está implementada por la clase "ActionSupport" por lo que estaría disponible sin necesidad de implementarla.

Como he explicado anteriormente cuando la validación es erronea los interceptores devuelven un result con nombre "input" por lo que hay que crear este Result en aquellos Actions sobre los que se realice validación en alguno de sus campos. Cuando se hace la validación lo normal es que se quiera mostrar en el JSP cual ha sido el fallo, para ello hay que incluir en el JSP oportuno las siguientes etiquetas:

```
<s:fielderror />
<s:actionerror />
```

La primera etiqueta sirve para mostrar los errores de validación que se den en la validación individual de los campos mientras que la segunda etiqueta muestra los errores que se dan en las validaciones planas. Hay que tener cuidado con el tipo de Result que pongamos a los interceptores de validación ya que si ponemos por ejemplo el Result de "redirectAction" se pierden los valores al crearse una petición nueva.

A la vista de los tres métodos de validación he decidido utilizar el método de validación mediante archivos XML [9] porque me permite desacoplarlo de la clase Action y en el caso de que necesite cambiar alguna validación no necesitaría recompilar la clase. Un ejemplo de una validación en la aplicación sería:

```
<field name="mail">
<field-validator type="email">
<message>El mail no tiene un formato valido</message>
</field-validator>
<field-validator type="requiredstring">
<message>El mail no puede estar vacío</message>
</field-validator>
<field-validator type="stringlength">
<param name="maxLength">45</param>
<message>El tamaño máximo del mail tiene que ser de 45</message>
</field-validator>
</field>
```

Esta validación pertenece a "LoginAction" y en ella compruebo que el mail tenga un formato válido, que esté informado y que su longitud máxima sea de 45 carácteres. Como se puede ver en cada validación incluyo un mensaje que es el que se muestra en caso de error. Un ejemplo de una validación plana en la aplicación sería:

```
<validator type="expression">
<param name="expression">pass.equals(pass1)</param>
```

```
<message>Las contraseñas tienen que ser iguales</message>
</validator>
```

Como se puede ver compruebo que las dos contraseñas sean iguales y en caso de que no lo sean muestro un mensaje de error.

4.3.2. Diseño del modelo

Gestión de usuarios

En este paquete se delegan todas las funciones de la aplicación encargadas de los usuarios. La interfaz de este paquete se llama "IGestionUsuarios", la mayoría de la funcionalidad de este paquete se delega en la capa de persistencia ya que la casi totalidad de la funcionalidad de los usuarios pasa por las siguientes funciones:

- Registro de usuarios en la aplicación
- Modificación de los datos que hay almacenados de un usuario
- Eliminación de toda la información de que dispone la aplicación sobre el usuario como es la información propia del usuario y las búsquedas que ha llevado a cabo
- Comprobar que las credenciales introducidas sean las correctas

Como este paquete no es el que se comunica con el sistema gestor de la base de datos toda esa funcionalidad la delega en la capa de persistencia como se puede ver en el diagrama de clases y en los diagramas de secuencia. Una funcionalidad que si que se lleva a cabo en este paquete es la de calcular el hash criptográfico de la contraseña para almacenarlo en la base de datos, este punto se trata más adelante.

Gestión de productos

Este paquete tiene varias funciones:

- Por un lado se encarga de delegar las operaciones de recuperación de información sobre productos como puedan ser el obtener el histórico de precios de un producto o recuperar toda la información de un producto.
- También se encarga de generar las distintas búsquedas en las fuentes de datos y pasar la información obtenida a la capa de persistencia
- Otra función es la de refrescar la información que hay en la aplicación periódicamente.

Como la obtención de los datos se va a hacer mediante la interfaz web que proporcionan las empresas tenía que utilizar un componente que hiciera de navegador web para permitirme programáticamente ir accediendo a las distintas páginas. Estos componentes se conocen como Webdriver, aunque la finalidad principal de un webdriver sea la de automatizar pruebas sobre páginas webs para hacer menos pesada esa tarea, estos componentes se pueden utilizar para el web scraping. Estas herramientas permiten al desarrollador simular el comportamiento de un navegador web: presionar botones, obtener el código HTML, cargar contenido dinámico mediante AJAX y todo ello con la potencia que da el usarlo detrás de un lenguaje de programación. Como webdriver utilicé HtmlUnit [10], este navegador web provee de una API para invocar páginas, rellenar formularios, hacer click en enlaces, etc. Una de las desventajas de utilizar un webdriver (sea cual sea) es su lentitud por lo que se ha intentado a toda costa reducir el número de páginas que se tienen que cargar.

Una vez obtenidas las páginas se podría acceder a su contenido mediante programación y tratamiento de cadenas pero hay formas más avanzas (y flexibles) de hacer esta tarea. Para esta parte utilicé la herramienta JSoup [11] que permite parsear páginas HTML desde una URL, un archivo o un string, encontrar y extraer datos utilizando selectores CSS, manipular los elementos HTML, etc. Aunque tiene muchas funciones yo lo voy a utilizar solamente para parsear HTML, es cierto que esto podría hacerlo en el webdriver ya que acepta la obtención de elementos HTML mediante XPath pero el utilizar un parser específico me parece más flexible.

La parte más importante de este paquete (y probablemente de toda la aplicación) es la de los wrappers , he creado cuatro y se encargan tanto de obtener la información "bruta" de las fuentes de datos como de tratar esa información para recoger los datos útiles. Los wrappers son los componentes de un sistema de integración de la información que se comunican con las fuentes de datos. Entre las tareas de un wrapper está el ir hasta las fuentes de datos, obtener los datos, transformarlos si es necesario,etc. El obtener información de las fuentes de datos no tiene por qué ser algo implicitamente complejo, un wrapper podría ser un programa que obtuviera datos de una base de datos relacional o incluso de un fichero de texto.

En el contexto de esta aplicación cada página web se considera una fuente de datos S que muestra datos estructurados utilizando un esquema de origen T_S y un formato F_S que son comúnes a todas las páginas web de S. Con las definiciones anteriores se podría decir que un wrapper W se encarga de extraer información estructurada de S. El wrapper se encarga por tanto de extraer la información utilizando la información que conoce de la fuente de datos y la organiza según un esquema de destino T_W con un formato de destino F_W. Sobra decir que ni T_S tiene que ser igual a T_W ni F_S a F_W. Otra de las funciones realizadas por el wrapper es la de formatear los datos de origen para que se adapten al formato de destino.

En la aplicación que he desarrollado he decidido hacer una construcción manual de los wrappers debido a varios motivos siendo el principal que el número de fuentes de datos era razonable y quería una aplicación que se pudiera mantener de forma fácil y para ello la mejor opción es la creación manual de un wrapper. Esta forma de crear un wrapper es la más obvia de todas ya que consiste en que una persona "observe" las fuentes de datos hasta conocer muy bien tanto el

esquema de origen como el formato de origen. Una vez conseguido ese punto tiene que construir un programa de extracción para obtener todos los datos y convertirlos al esquema de destino que desee con el formato de destino que necesite. Dentro de esta forma de crear un wrapper hay dos aproximaciones a la hora de extraer la información de la página web:

- Se puede obtener la información tratando la cadena de texto (el código HTML) como si fuera un conjunto de palabras.
- La otra alternativa es considerar la cadena de texto como un árbol DOM e ir navegándolo hasta obtener la parte de la página web que se desee. Hay dos lenguajes de consulta muy importantes para lograr esto como son XPath y CSSQuery.

Para la aplicación que he creado he optado por aplicar un método de parsing basado en CSSQuery, podría haber utilizado XPath pero me parecen mucho más fácil de entender las expresiones CSSQuery, más sencillas de mantener y más cortas. Por ejemplo las siguientes expresiones que obtienen una imagen de una página web son equivalentes:

- XPath

  ```
  /html/body/div[15]/div/div/div[3]/a/img
  ```

- CSSQuery

  ```
  body > div:nth-child(15) > div > div > div.image_line.no_precio_ant > a > img
  ```

Como se puede ver la segunda expresión da más información, es más natural y depende menos de la distribución de la página.

Uno de las ventajas de este método es que estos wrappers son bastante fáciles de entender y de mantener. Por otro lado tiene varias desventajas como son que esta opción no es muy escalable ya que necesita un gran esfuerzo y otra de las desventajas es que un cambio en la interfaz de las fuentes de datos podría provocar que los wrappers dejaran de obtener los datos correctamente.

La arquitectura general de estos wrappers es la siguiente:

- IWrapper: Es la interfaz mediante la cual el cliente ejecuta el wrapper, gracias a esto se ha intentado minimizar el acoplamiento con las capas superiores. Al utilizar simplemente una interfaz es muy sencillo añadir o eliminar wrappers a la aplicación.
- Wrapper: Es la clase que implementa la interfaz anterior y su única funcionalidad es la de llamar a la clase encargada de la obtención de las páginas. Se ha decidido crear esta clase en vez de hacer una llamada directa a la clase de obtener páginas por si en un futuro se desea añadir alguna funcionalidad en este punto y para que cada clase se ocupe solamente de sus tareas.
- ObtenerPaginas: Esta clase es la encargada de la reformulación de las consultas y de comunicarse con las fuentes de datos para obtener el código HTML del que se obtendrán los productos.

- Parsing: Esta clase es llamada por la clase anterior y su función es obtener información de las páginas. Esa información puede ser tanto obtener los productos como obtener cualquier parámetro de la página. Una vez ha terminado su función llamada al integrador para que se encargue de insertar los productos.

Esta arquitectura de los wrappers sigue el esquema ETL ya que en resumidas cuentas estos componentes extraen la información de las fuentes de datos, la transforman (el formato cuando es necesario) y la cargan en el warehouse.

Wrapper 2 Este wrapper se encarga de obtener las pagina del supermercado Eroski [12]. Para obtener productos de esta página mediante la interfaz web no hay ningún filtro de login ni código postal. El diagrama de clases de este wrapper es el siguiente:

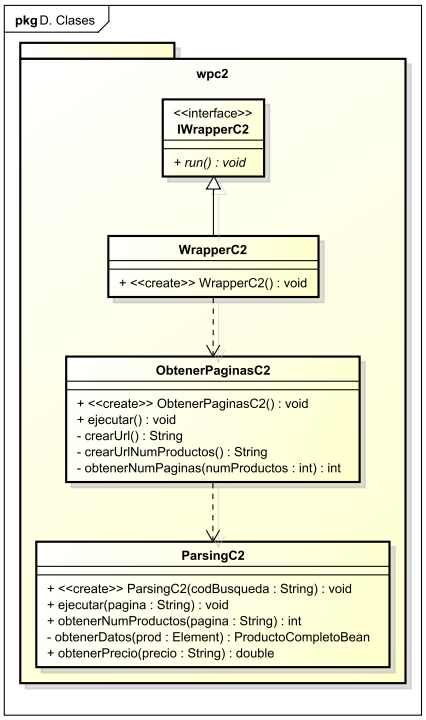

Figura 20: Diagrama de clases del Wrapper 2

- URLs utilizadas

  - URL1: Para obtener la primera página de productos utilizo la URL que se puede ver en el navegador.

    `http://www.compraonline.grupoeroski.com/supermercado/buscador.jsp?q=producto`

    Esta url admite solo un parámetro que es el nombre del producto, en caso de haber espacios en el nombre del producto se sustituyen por " %20".

  - URL2: Para ir obteniendo el resto de productos utilizo la siguiente url:

    `http://www.compraonline.grupoeroski.com/supermercado/ajax/listProductsBuscador.jsp?q=producto&productsPerPage=productosPorPagina&page=numPagina`

A esta url se le pasan los siguientes parámetros:

- q: Al igual que en el caso anterior, este parámetro sirve para decir qué producto vamos a buscar, en caso de que haya espacios en el producto se substituyen por "
- productsPerPage: Aquí se indica el número de productos por página.
- page: En este último indicamos qué número de página es (en este caso el número de página se identifica con la cantidad de veces que el usuario ha hecho scroll down ya que en esta página no hay una paginación clásica).

- Obtención de páginas

Los pasos seguidos son los siguientes:

1. Obtengo la primera página de resultados del producto utilizando la URL1.

```
HtmlPage pag1 = this.webClient.getPage(URL1);
```

2. Sobre esta página ejecuto una función javascript para que se terminen de cargar la página los filtros de este producto.

```
String jsCode = "initialLoad()";
ScriptResult result = pag1.executeJavaScript(jsCode);
HtmlPage pag2 = (HtmlPage) result.getNewPage();
```

3. Esta página que acabo de obtener la utilizo para recorrer los filtros del producto y así obtener el número total de productos y a partir de ese número total de productos calculo el número total de páginas.

```
int numProductos = pc2.obtenerNumProductos(pag2.asXml());
int numPaginas = obtenerNumPaginas(numProductos);
```

4. Cuando he obtenido el número total de páginas voy iterando y en cada iteración modificado la url2 para que se adapte en cada caso.

- Programa de extracción

1. Distribución gráfica

Figura 21: Distribución gráfica de los elementos en la fuente de datos 2

2. Esquema de origen: Como se puede ver los productos están organizados en un div cada uno directamente dentro del ”body“ del código html.

Figura 22: Esquema de origen de la fuente de datos 2

3. Obtención de productos: Para obtener obtener todos los productos cojo todos aquellos que tengan la clase “top_element”. Como se puede ver podría haber codigo aquellos elementos que tuvieran las clases “module”, “mod1col”, “product_mod” pero lo único que hubiera conseguido sería haber tenido que seleccionar despues aquellos con la clase “top_element”. Para obtener esos productos utilizo:

```
Elements elementos = doc.select(".top_element");
```

4. Obtención de datos: Una vez obtenidos todos los productos en el paso anterior queda recorrer la lista de productos para obtener los datos que interesan, la información de cada producto está organizada de la siguiente forma:

Figura 23: Distribución y formato de los datos en la fuente de datos 2

Como se puede ver la información está repartida de forma bastante uniforme. Los datos que obtengo son los siguientes:

- Identificador del producto

  ```
  String id = prod.select(".add_box.prod_carrito").attr("idProducto");
  ```

- Precio

  ```
  String precio = prod.select(".precio_line").text();
  ```

- Precio por cantidad

  ```
  String precioPorCantidad = prod.select(".description_2").text();
  ```

- Descripción

  ```
  String descripcion = prod.select(".description_1").text();
  ```

- Url de la imagen

  ```
  String urlImagen = prod.select(".image_line.no_precio_ant").
  select(".iframelayer_ficha.cboxElement").select("img").
  first().attr("src");
  ```

- Url del producto

  ```
  String urlProducto =  prod.select(".iframelayer_ficha").attr("href");
  ```

5. Formato de origen de los datos

   - Identificador del producto: Cadena de texto limpia que contiene el identificador del producto.
   - Precio: Cadena de texto, la parte entera y la parte decimal está separada por una coma y al final contiene el simbolo del euro.
   - Precio por cantidad: Cadena de texto.
   - Descripción: Cadena de texto.
   - Url de la imagen: Cadena de texto.
   - Url del producto: Cadena de texto

6. Formato de destino de los datos
    - Identificador del producto: Cadena de texto.
    - Precio: Tiene que ser un número decimal, la parte entera y la parte decimal tienen que estar separadas por un punto y no tiene que haber ningún otro símbolo.
    - Precio por cantidad: Cadena de texto.
    - Descripción: Cadena de texto.
    - Url de la imagen: Cadena de texto.
    - Url del producto: Cadena de texto.
7. Las transformaciones necesarias
    - Precio: Se mantienen todos los números del precio y la coma es reemplazada por un punto.

Wrapper 3 Este wrapper se encarga de obtener las pagina del supermercado Lupa [13]. En esta página web existe un mecanismo por el cual hay que pasar previamente por una página web en la que hay que insertar un código postal y presionar un botón (estas dos acciones se llevan a cabo mediante mecanismos del webdriver). El diagrama de clases de este wrapper es el siguiente:

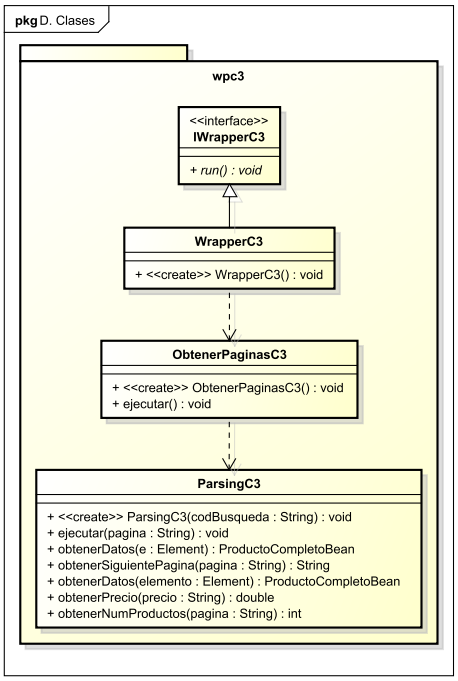

Figura 24: Diagrama de clases del Wrapper 3

- URLs utilizadas
    - URL_PRINCIPAL: Es la url a partir de la cual genero el resto.

      `http://www.lupaonline.com/`

- Obtención de páginas

  Los pasos seguidos son los siguientes:

1. El primer paso es obtener la página donde tengo que insertar el código postal:

   ```
   HtmlPage pag1 = this.webClient.getPage(URL_PRINCIPAL);
   ```

2. Esta página es la principal del supermercado, una vez que la tengo tengo que introducir en el campo de texto correspondiente el valor del código postal:

   ```
   HtmlForm formCP = pag1.getFormByName("cp-form");
   HtmlTextInput textFieldCP = (HtmlTextInput) pag1.getElementById("cp");
   textFieldCP.setValueAttribute("47012");
   ```

3. Cuando se ha introducido el código postal lo más rápido y más flexible es crear un botón en esa página y añadirselo al formulario para que se pueda hacer submit:

   ```
   HtmlElement buttonCP = (HtmlElement) pag1.createElement("button");
   buttonCP.setAttribute("type", "submit");
   formCP.appendChild(buttonCP);
   HtmlPage pag2 = buttonCP.click();
   ```

4. La página que se obtiene tras pulsar ese botón es en la que se introduce el producto, el mecanismo es similar al anterior, hay que recuperar el formulario que se utiliza para la búsqueda:

   ```
   HtmlForm formProductos = (HtmlForm) pag2.getElementById("search_mini_form");
   HtmlTextInput textFieldProductos = formProductos.getInputByName("q");
   textFieldProductos.setValueAttribute(cadena);
   ```

5. De nuevo creo un botón para enviar los datos al servidor:

   ```
   HtmlElement buttonProductos = (HtmlElement) pag2.createElement("button");
   buttonCP.setAttribute("type", "submit");
   formProductos.appendChild(buttonProductos);
   HtmlPage pag3 = buttonProductos.click();
   ```

6. La página que se obtiene tras pulsar el botón de productos contiene el primer grupo de productos, para seguir recuperando páginas utilizo el mismo mecanismo que utilizaría un usuario, dar al botón de siguiente página. Para ello recupero el botón de"Siguiente página" de la página actual y sigo hasta que no haya botón de siguiente página.

- Programa de extracción

1. Distribución gráfica

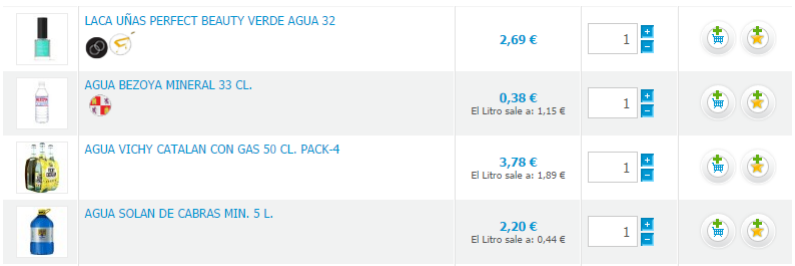

Figura 25: Distribución gráfica de los elementos en la fuente de datos 3

2. Esquema de origen: Todos los productos están dentro de una tabla y cada uno de ellos ocupa una fila de dicha tabla.

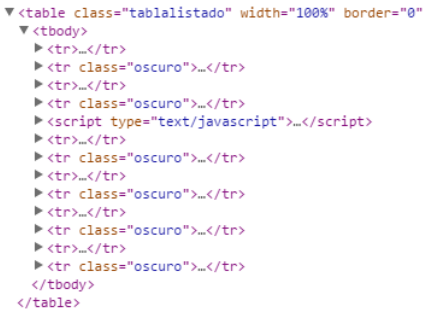

Figura 26: Esquema de origen de la fuente de datos 3

3. Obtención de productos: Para obtener todos los productos me quedo con la tabla que tiene la clase “tablalistado”, de esa tabla cojo su hijo directo “tbody” y de ese elemento sus hijos de tipo “tr”:

```
Elements listaTR = doc.select("table[class=tablalistado] > tbody > tr");
```

4. Obtención de datos: Una vez obtenidos todos los productos en el paso anterior queda recorrer la lista de productos para obtener los datos que interesan, en esta fuente de datos me he encontrado con que hay dos supuestos la única diferencia está en cómo se organiza el precio:

Caso 1

Figura 27: Distribución y formato de los datos en la fuente de datos 3

Caso 2

Figura 28: Distribución y formato de los datos en la fuente de datos 3

Los datos que obtengo son los siguientes:

- Identificador del producto

  ```
  String codProducto = elemento.select("h6").first().attr("id").
  replace("precio_unidad-", "");
  ```

- Precio: Como para este dato había dos casos tengo que utilizar dos formas diferentes
  - Caso 1

    ```
    String precio = elemento.select("h5").first().text();
    ```

  - Caso 2

    ```
    String precio = e.select("p.special-price > span.price").text();
    ```

- Precio por cantidad

  ```
  String precioPorCantidad = elemento.select("h6").first().text();
  ```

- Descripción

  ```
  String descripcion = elemento.select("h4").first().text();
  ```

- Url de la imagen

  ```
  String urlImagen = elemento.select("div.cubreimglista > a").
  select("img").first().
  absUrl("src");
  ```

- Url del producto

  ```
  String urlProducto = elemento.select("div.cubreimglista > a").
  attr("href");
  ```

5. Formato de los datos obtenidos
   - Identificador del producto: El identificador del producto está formado por el identificador y por la cadena “precio_unidad-”.
   - Precio: Cadena de texto, en el primer caso la parte entera y la parte decimal está separada por una coma y al final contiene el simbolo del euro mientras que en el

segundo caso la parte entera y la parte decimal están separadas por una coma y al final contiene el símbolo del euro y la cadena "/kg".

- Precio por cantidad: Cadena de texto.
- Descripción: Cadena de texto.
- Url de la imagen: Cadena de texto.
- Url del producto: Cadena de texto

6. Formato de destino de los datos

- Identificador del producto: Cadena de texto.
- Precio: Tiene que ser un número decimal, la parte entera y la parte decimal tienen que estar separadas por un punto y no tiene que haber ningún otro simbolo.
- Precio por cantidad: Cadena de texto.
- Descripción: Cadena de texto.
- Url de la imagen: Cadena de texto.
- Url del producto: Cadena de texto.

7. Transformaciones necesarias son las siguientes:

- Identificador del producto: La transformación como es muy simple se realiza al mismo tiempo que se obtiene el identificador reemplazando la cadena que sobra por una cadena vacía.
- Precio: Se mantienen todos los números del precio y la coma es reemplazada por un punto.

Wrapper 4 Este wrapper se encarga de obtener las pagina del supermercado Gadis [14]. Para acceder a los productos de este supermercado hay que pasar por un paso previo en el que se introduce el código postal para comprobar si existe reparto en esa zona, como la finalidad de la aplicación es mostrar los productos introduzco un código postal sobre el que sé que hay reparto. El diagrama de clases de este wrapper es el siguiente:

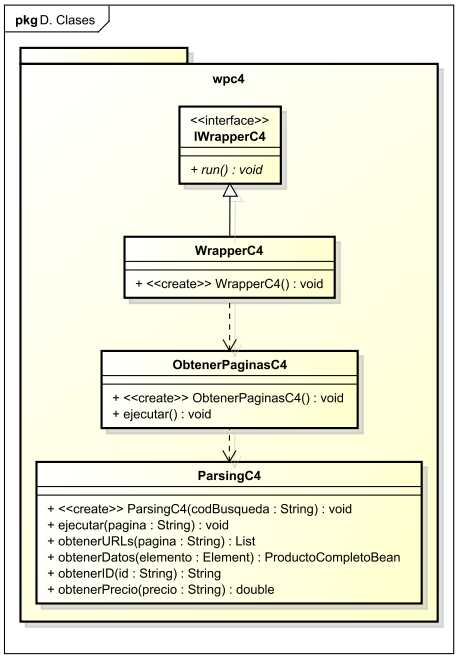

Figura 29: Diagrama de clases del Wrapper 4

- URLs utilizadas

  - URL_PRINCIPAL: Es la url principal del supermercado y a partir de la cual genero el resto.

    `http://www.gadisline.com/`

- Obtención de páginas

  Los pasos seguidos son los siguientes:

  1. El primer paso es obtener la página donde tengo que insertar el código postal:

     ```
     HtmlPage pag1 = this.webClient.getPage(URL_PRINCIAL+"web/gadisa.jsp");
     ```

  2. En esta página tengo que coger el formulario donde se insertar el código postal y añadir el valor:

     ```
     HtmlForm formCP = pag1.getFormByName("formVisitanteCP");
     HtmlTextInput textFieldCP = formCP.getInputByName("codigoPostal");
     textFieldCP.setValueAttribute("47012");
     ```

  3. Una vez introducido el código postal creo un botón en esa página y lo asocio al formulario:

     ```
     HtmlElement buttonCP = (HtmlElement) pag1.createElement("button");
     buttonCP.setAttribute("type", "submit");
     formCP.appendChild(buttonCP);
     buttonCP.click();
     ```

4. Al hacer click en ese botón consigo que la página web me genere una cookie en el navegador del web driver y ahora puedo acceder mediante URL. El siguiente paso es introducir el producto en la página correspondiente, la página donde se introduce está formada por varios frames de jsp por lo que me vale con cargar el frame que contiene el formulario:

```
HtmlPage pag2 = this.webClient.getPage(URL_PRINCIPAL+"web/buscar.jsp");
```

5. En esta página obtengo el formulario donde se inserta la cadena a buscar y también el campo de texto donde introduzco la cadena porque no está dentro del formulario.

```
HtmlForm formProducto = pag2.getFormByName("formulario");
HtmlTextInput textFieldProducto = pag2.getElementByName("fraseBusqueda");
textFieldProducto.setValueAttribute(cadena);
```

6. Genero un botón dentro del formulario y asocio el campo de texto obtenido en el paso anterior a ese formulario:

```
HtmlElement buttonProducto = (HtmlElement) pag2.createElement("button");
buttonProducto.setAttribute("type", "submit");
formProducto.appendChild(buttonProducto);
formProducto.appendChild(textFieldProducto);
HtmlPage pag3 = buttonProducto.click();
```

7. Esta página obtenida tras pulsar el botón contiene el primer conjunto de resultados, el siguiente paso es recoger todas las urls que se pueden obtener.

```
List<String> urls = pc4.obtenerURLs(pag3.asXml());
```

8. Por último voy recorriendo esta lista hasta haber obtenido todas las URLs

- Programa de extracción

1. Distribución gráfica

Figura 30: Distribución gráfica de los elementos en la fuente de datos 4

2. Esquema de origen: En este caso los productos se encuentran distribuidos en una tabla y cada uno de ellos ocupa una fila de esa tabla.

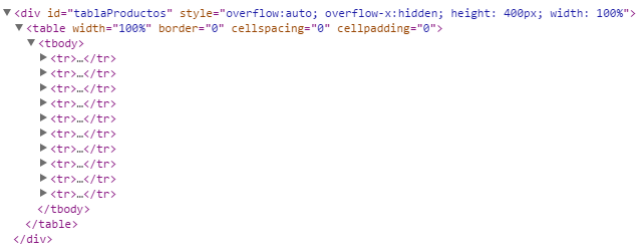

Figura 31: Esquema de origen de la fuente de datos 4

3. Obtención de productos: Para obtener los productos de esta página me quedo con el div que tiene el identificador "tablaProductos", dentro de el con la tabla, luego con el body de esa tabla y por último con todas las filas:

   ```
   Elements listaTR = doc.select("div#tablaProductos > table > tbody > tr");
   ```

4. Obtención de datos: Una vez obtenidos todos los productos en el paso anterior queda recorrer la lista de productos para obtener los datos que interesan, la información de cada producto está organizada de la siguiente forma:

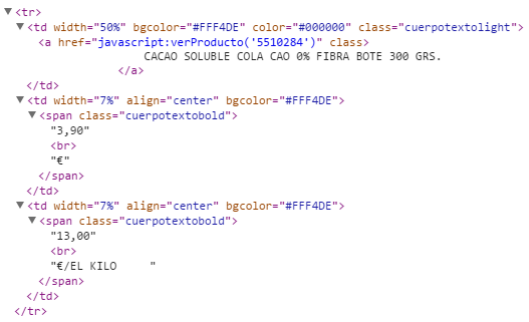

Figura 32: Distribución y formato de los datos en la fuente de datos 4

   Los datos que obtengo son los siguientes:

   - Identificador del producto

     ```
     String id = td.get(1).select("a").attr("href");
     ```

   - Precio

     ```
     String precio = td.get(4).select("span").text();
     ```

   - Precio por cantidad

     ```
     String precioPorCantidad = td.get(5).select("span").text();
     ```

   - Descripción

     ```
     String descripcion = td.get(1).select("a").text();
     ```

5. Formato de origen de los datos obtenidos

   - Identificador del producto: La cadena de texto de donde obtengo el identificador del producto tiene la siguiente forma:

     ```
     javascript:verPRoducto('idProducto')
     ```

   - Precio: Cadena de texto, la parte entera y la parte decimal está separada por una coma y al final contiene el simbolo del euro.

- Precio por cantidad: Cadena de texto.
- Descripción: Cadena de texto.

6. Formato de destino de los datos obtenidos:
   - Identificador del producto: Cadena de texto.
   - Precio: Tiene que ser un número decimal, la parte entera y la parte decimal tienen que estar separadas por un punto y no tiene que haber ningún otro simbolo.
   - Precio por cantidad: Cadena de texto.
   - Descripción: Cadena de texto.
7. Transformaciones necesarias
   - Identificador del producto: Para obtener el identificador del producto tal y como lo necesito voy recorriendo la cadena que tengo y me quedo solo con los caracteres que sean un número.
   - Precio: Se mantienen todos los números del precio y la coma es reemplazada por un punto.

Wrapper 5 Este wrapper se encarga de obtener las pagina del supermercado Alcampo [15]. No se puede acceder directamente al buscador y hay que pasar antes por una página donde hay que introducir un código postal para comprobar que exista reparto en esa zona, introduzco un código postal sobre el que se que hay reparto. El diagrama de clases de este wrapper es el siguiente:

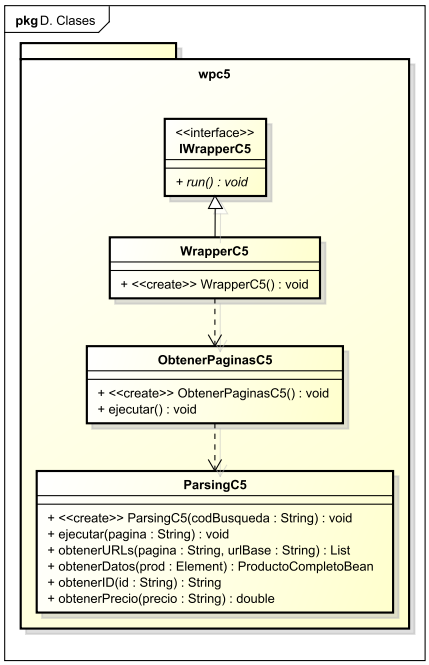

Figura 33: Diagrama de clases del Wrapper 5

- URLs utilizadas
  - URL_PRINCIPAL: Es la url principal del supermercado y a partir de la cual genero el resto.

    `https://www.alimentacion.alcampo.es/`

- Obtención de páginas

  Los pasos seguidos son los siguientes:

  1. Obtengo la página donde tengo que insertar el código postal.

     ```
     HtmlPage pag1 = this.webClient.getPage(URL_PRINCIPAL+"tienda/home.php");
     ```

  2. De esa página tengo que coger el formulario donde introducir el código postal.

     ```
     HtmlForm formCP = pag1.getFormByName("form");
     HtmlTextInput textFieldCP = formCP.getInputByName("cp");
     textFieldCP.setValueAttribute("09001");
     ```

  3. El paso siguiente es crear un botóndentro del formulario para poder hacer submit

     ```
     HtmlElement buttonCP = (HtmlElement) pag1.createElement("button");
     buttonCP.setAttribute("type", "submit");
     formCP.appendChild(buttonCP);
     HtmlPage pag2 = buttonCP.click();
     ```

  4. En esta página obtenida ya puedo insertar la cadena de texto, para ello lo primero que hago es recuperar el formulario:

     ```
     HtmlForm formProductos = pag2.getFormByName("quick_find");
     HtmlTextInput textFieldProductos = formProductos.
     getInputByName("keywords");
     textFieldProductos.setValueAttribute(cadena);
     ```

  5. Una vez recuperado el formulario e insertada la cadena creo un botón para hacer submit.

     ```
     HtmlElement buttonProductos = (HtmlElement) pag2.createElement("button");
     buttonProductos.setAttribute("type", "submit");
     formProductos.appendChild(buttonProductos);
     HtmlPage pag3 = buttonProductos.click();
     ```

  6. Esa página ya obtiene el primer grupo de productos, obtengo el total de urls del paginador y voy recorriendo para ir recuperando productos.

- Programa de extracción

  1. Distribución gráfica

Figura 34: Distribución gráfica de los elementos en la fuente de datos 5

2. Esquema de origen: Los productos están distribuidos como items de una lista no ordenada.

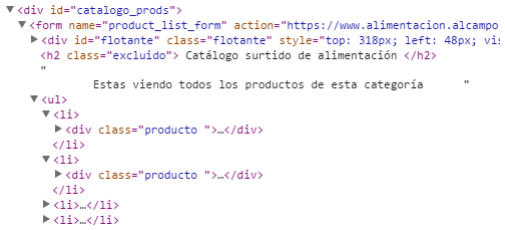

Figura 35: Esquema de origen de la fuente de datos 5

3. Obtención de productos: Para obtener todos los productos me quedo con el elemento que tiene el identificador “catalogo_prods” y dentro de el todos sus hijos que sean un div y tengan la clase “producto”.

```
Elements productos = doc.select("#catalogo_prods div.producto");
```

4. Obtención de datos: Una vez obtenidos todos los productos en el paso anterior queda recorrer la lista de productos para obtener los datos que interesan, la información de cada producto está organizada de la siguiente forma:

Figura 36: Distribución y formato de los datos en la fuente de datos 5

Los datos que obtengo son los siguientes:

- Identificador del producto

  ```
  String id = prod.select("div.info > div.price_and_buy > div.buy > div.cantidad
  div.cantidad > input[name=products_id").val();
  ```

- Precio

  ```
  String precio = prod.select("div.precio").text();
  ```

- Descripción

  ```
  String descripcion = prod.select("div.descripcion").text();
  ```

- Url de la imagen

  ```
  String urlImagen = prod.select("div.img_product > img").attr("src");
  ```

- Url del producto

  ```
  String urlProducto = prod.select("div.transparent_img > a.Ntooltip").attr("hre
  ```

5. Formato de origen de los datos obtenidos
   - Identificador del producto: Cadena de texto limpia que contiene el identificador del producto.
   - Precio: Cadena de texto, la parte entera y la parte decimal está separada por una coma y al final contiene el simbolo del euro.
   - Descripción: Cadena de texto.
   - Url de la imagen: Cadena de texto.
   - Url del producto: Cadena de texto
6. Formato de destino de los datos obtenidos
   - Identificador del producto: Cadena de texto.
   - Precio: Tiene que ser un número decimal, la parte entera y la parte decimal tienen que estar separadas por un punto y no tiene que haber ningún otro simbolo.

- Descripción: Cadena de texto.
- Url de la imagen: Cadena de texto.
- Url del producto: Cadena de texto.

7. Transformaciones necesarias de los datos
   - Precio: Se mantienen todos los números del precio y la coma es reemplazada por un punto.

Una vez que he explicado tanto las herramientas que utilizo en esta parte como la teoría de wrappers que he aplicado y los wrappers que he desarrollado solo queda saber cómo se une todo:

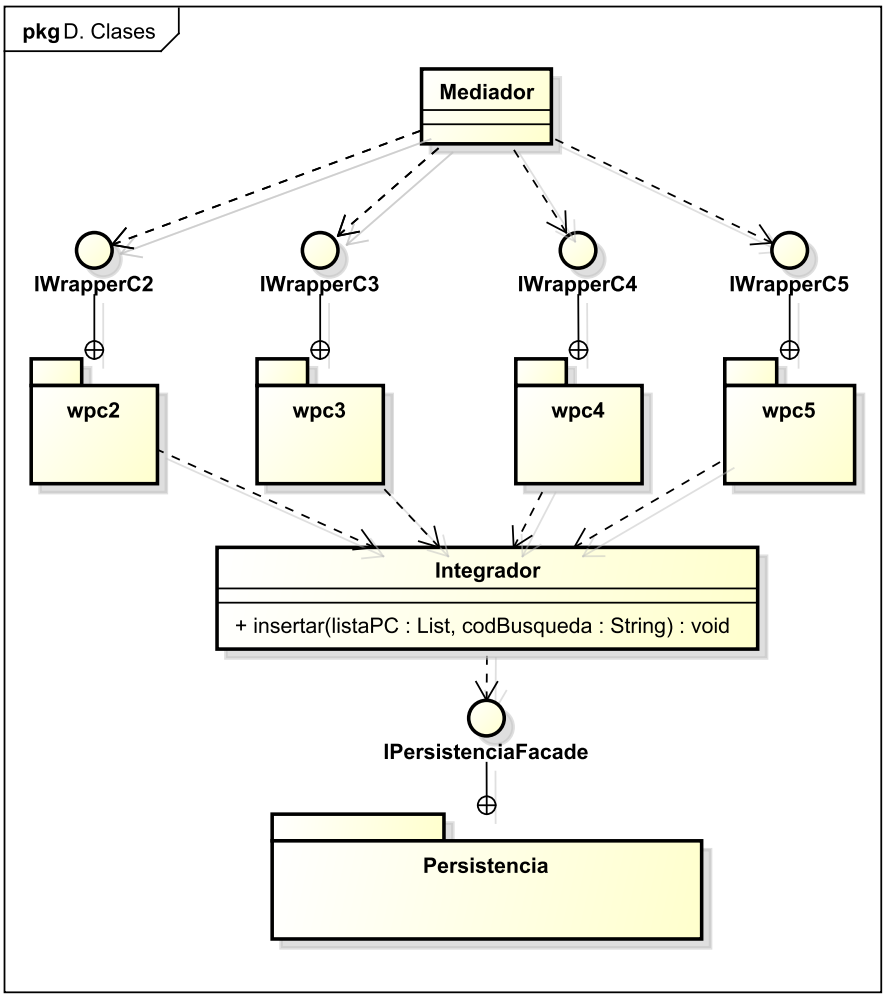

Figura 37: Diagrama de clases de los wrappers y el integrador

Por tanto cuando se va a lanzar una búsqueda sobre las fuentes de datos el mediador se encarga de llamar a los cuatro wrappers de forma asíncrona para que su ejecución se realice en paralelo. A medida que se van obtenindo las páginas se van parseando y la clase encargada de esta tarde se encarga de llamar al integrador. El integrador es la entidad encargada de, una vez obtenido el producto, ver si es válido, considero que un producto es válido cuando tiene los campos del código del producto, código del establecimiento, descripción y precio informados. Los objetos que manejan las clases encargadas del parsing contienen toda la información del producto. Estos objetos tienen que dividirse, por un lado se mantiene la información del producto, por otro

lado la información del precio de ese producto en ese instante y por último se crean objetos que relacionan la búsqueda a la que pertenece ese producto con el producto en si. Es este componente (el integrador) el que se encarga de proporcionar al mecanismo de persistencia los datos para que se introduzcan en la base de datos.

4.3.3. Diseño de la capa de persistencia

Arquitectura de la persistencia En la capa de persistencia se decidió utilizar MySQL como sistema gestor de base de datos. Como forma de acceso a datos voy a utilizar JDBC junto al patrón DAO. Este patron permite abstraer y encapsular el acceso a la fuente de datos y se encarga de gestionar la conexión con la fuente de datos para obtener y almacenar datos. Junto al patrón DAO voy a utilizar el patrón Abstract Factory para maximizar la flexibilidad de este patrón permitiendo así tener varias fuentes de datos sin que el cliente que utilice la persistencia note en ningún momento cual está por debajo [16].

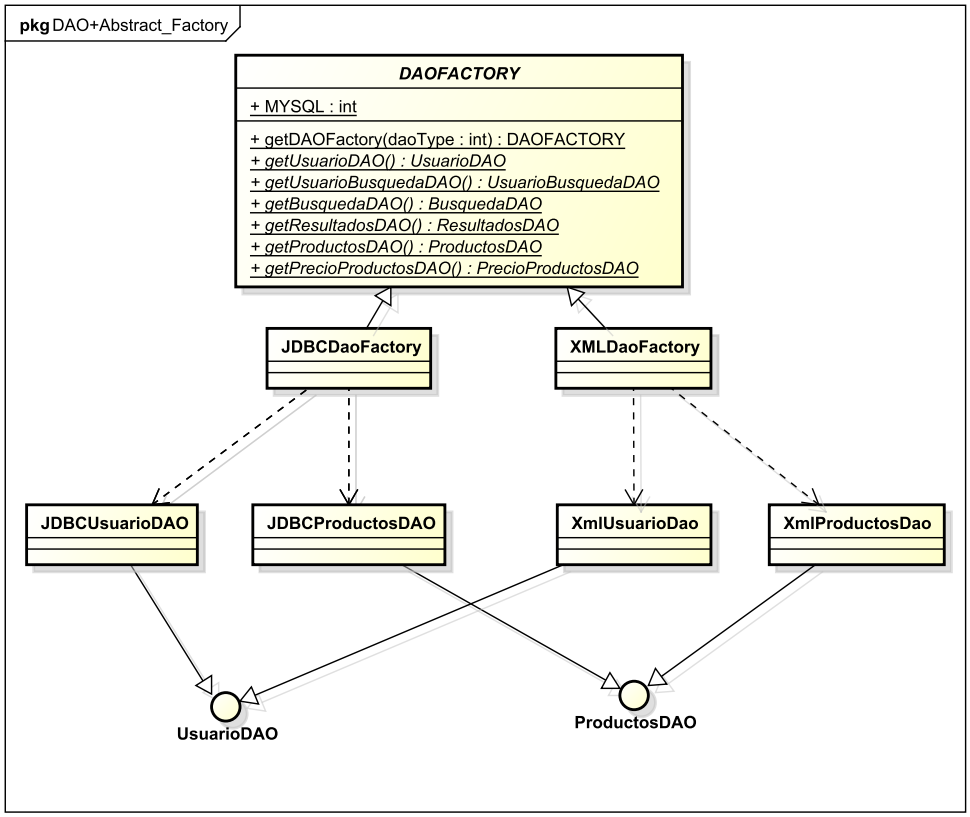

Figura 38: Diagrama de clases del patrón DAO y el patrón Abstract Factory

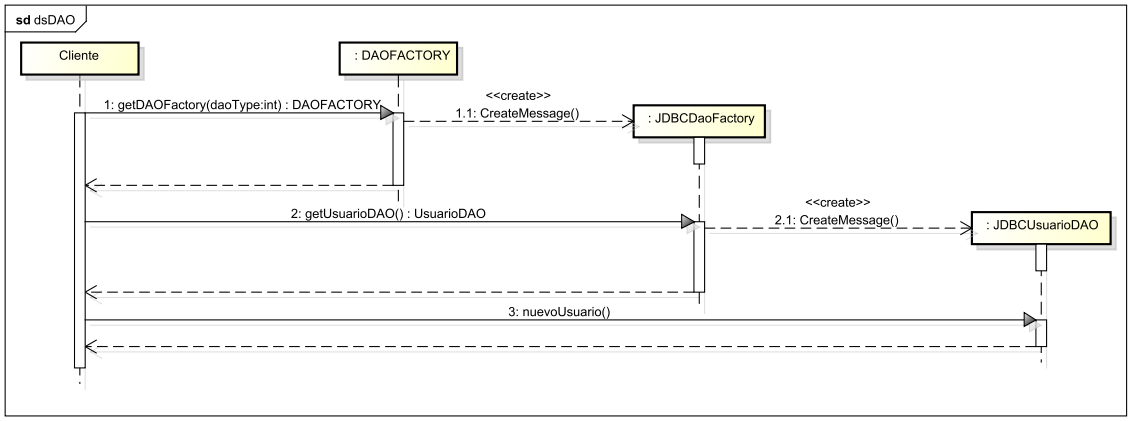

Figura 39: Diagrama de secuencia del patrón DAO y el patrón Abstract Factory

Como se puede ver en el diagrama de secuencia, se delega la creación del DAO al objeto DAOFACTORY y de este modo el cliente puede elegir el modo de acceso a datos, se podría restringir el patrón para que el usuario utilizara un único modo de acceso a datos sin dejarle elegir.

Framework para el Connection Pool Uno de los problemas al desarrollar una aplicación que tenga un componente de acceso a datos en el que ese acceso se realice a una base de datos es la gestión de la conexión. Cuando se está desarrollando una aplicación de este tipo y se hacen pruebas el impacto sobre la base de datos es muy pequeño y no aparece ningún problema pero en el momento en el que se pasa la aplicación a entornos de explotación vienen los problemas. El coste de abrir una conexión con la base de datos es muy alto y si estamos constantemente creando y destruyendo conexiones la fluidez de la aplicación va a dismininuir mucho. Por ello una de las técnicas más usadas es la del Connection Pool. La idea de esta técnica es la siguiente, en vez de crear y destruir conexiones constantemente lo que se hace es mantener una serie de conexiones a la base de datos abiertas y cuando se necesita una conexión se toma prestada una conexión de este pool y cuando se termina se devuelve a este pool de este modo un pequeño número de conexiones puede dar servicio a un gran número de solicitudes.

Una vez visto cual es el problema de la gestión de la conexión he decidido utilizar un framework que mantenga ese pool de conexiones y así poder utilizarlo de forma transparente. Entre los frameworks que había disponibles elegí C3P0 [17]. Uno de los motivos para elegir esta herramienta fue que su última versión es bastante actual [18] y que cumplía con todas mis necesidades de cara a esta aplicación. Este framework acepta muchos parámetros [19], algunos de los más útiles son:

- acquireIncrement: Determina cuantas conexiones va a intentar obtener C3P0 cuando el pool de conexiones se agote.
- driverClass: Nombre del driver que va a crear las conexiones.
- maxPoolSize: Es el número máximo de conexiones que van a estar disponibles en el pool.
- minPoolSize: Es el número mínimo de conexiones que van a estar disponibles en el pool.
- initialPoolSize: Número de las conexiones de las que se dispondra en el pool al inicio.

- maxStatements: Este parámetro controla el número de statements que van a estar en cache para **todas las conexiones**.

La configuración de C3P0 se puede hacer tanto en un fichero de propiedades como en un fichero XML, en la aplicación he utilizado un fichero de propiedades y los valores son los siguientes:

```
minPoolSize = 10
maxPoolSize = 75
maxStatements = 180
acquireIncrement = 3
driverClass = com.mysql.jdbc.Driver
user = root
password = root

testConnectionOnCheckout = false
testConnectionOnCheckin = true
idleConnectionTestPeriod = 30

jdbcUrl = jdbc:mysql://localhost:3306/TFG
```

Mantenimiento de la conexión Uno de los problemas al crear una aplicación que haga uso de conexiones con una base de datos es mantener "vivas" las conexiones. A pesar de que el problema sea que el sistema gestor de bases de datos crea una conexión por un período de tiempo hay que solucionar ese problema en este punto. Para ello desde la documentación oficial de C3P0 [20] recomiendan usar sus parámetros de configuración para que el propio framework compruebe cada cierto tiempo que las conexiones del pool están disponibles y listas para usar. Los parámetros que se utilizan para ello son:

```
testConnectionOnCheckout = false
testConnectionOnCheckin = true
idleConnectionTestPeriod = 30
```

Interfaz y excepciones Como punto de entrada a la funcionalidad de esta capa he creado una interfaz que es utilizada por la capa posterior (por la gestión de usuarios y por la gestión de productos). La funcionalidad de la clase que implementa esta interfaz es muy pequeña ya que su única función es obtener el DAO oportuno en cada caso e invocar el método de ese DAO.

Para gestionar los distintos errores o situaciones que se vayan dando en esta capa he creado varias excepciones:

- BusquedaExistente: Esta excepción se lanza desde la capa de persistencia cuando se detecta que se intenta insertar una cadena de una búsqueda que ya existía.
- CadenaNoEncontrada: Esta excepción se lanza desde la capa de persistencia cuando se quiere saber si existía una búsqueda asociada a una determinada cadena.

- LoginIncorrecto: Esta excepción se lanza cuando las credenciales introducidas por el usuario no son correctas.
- UsuarioYaExistente: Esta excepción se utiliza para determinar cuando está en uso el loggin de un usuario, ya sea porque se está registrando o porque lo quiere modificar.
- UsuarioNoEncontrado: Esta excepción es lanzada por la capa de persistencia cuando se va a hacer una búsqueda sobre un usuario y no se encuentra información de él.

Operaciones atómicas En esta aplicación en algunos casos he tenido la necesidad de realizar operaciones atómicas para que si ocurria algún error intermedio no se hubiera producido ningún cambio en la base de datos. Para ello he creado una clase que se llama "OperacionesCompuestas" en la cual están las operaciones que requieren esas características. Para lograr este efecto de atomicidad el procedimiento que llevo a cabo es el siguiente:

1. Obtengo una conexión del pool
2. Deshabilito el autocommit de esa conexión
3. Utilizo esa conexión para realizar las operaciones pertinentes, utilizo **la misma conexión** con todas las operaciones.
4. Al terminar de realizar las operaciones hago un commit en esa conexión.

Si se produce cualquier error durante la ejecución de algunas de las operaciones se hace rollback y la base de datos queda en un estado anterior a cualquier operación. Se produzcan o no errores antes de devolver la conexión al pool vuelvo a habilitar el autocommit para que la próxima vez que se vuelva a utilizar esa conexión lo tenga habilitado.

4.3.4. Diseño de la BBDD

Requisitos sobre los datos

Requisitos de los usuarios

- El sistema debe asegurar que no haya dos usuarios con el mismo loggin.
- El sistema debe permitir el almacenamiento del hash de la contraseña del usuario con un tamaño de 128 posiciones.
- El sistema debe permitir almacenar el salt con el que se crea el hash de la contraseña del usuario.

Requisitos de las búsquedas de un usuario

- El sistema debe poder identificar cada una de las búsquedas que se lleven a cabo en el sistema registrando qué usuario la ha hecho y en qué instante de tiempo.
- El sistema debe poder registrar cuando ha terminado cada uno de los wrappers para esa búsqueda de ese usuario.

- El sistema debe poder distinguir cuando una búsqueda de un usuario se ha hecho contra las fuentes de datos o cuando se ha hecho contra la base de datos propia.

Requisitos de las búsquedas

- El sistema deberá asegurar que no se almacena repetida la misma búsqueda.
- El sistema deberá mantener la última fecha de obtención de cada búsqueda.
- El sistema deberá poder almacenar la cadena que pertenezca a una búsqueda hasta una longitud de 45 carácteres.

Requisitos de los resultados

- El sistema debe permitir que se identifique cuándo un producto pertenece a una fuente de datos y a una búsqueda.

Requisitos de los productos

- El sistema deberá mantener la última fecha de obtención de cada producto.
- El sistema deberá identificar cada producto a lo largo de todos los establecimientos que haya.
- El sistema deberá permitir almacenar las URLs de los productos hasta una longitud máxima de 200 carácteres.
- El sistema deberá permitir almacenas las URLs con las que se acceden a las imágenes de los productos hasta una longitud máxima de 300 carácteres.
- El sistema deberá almacenar la última fecha en la que se ha obtenido un producto con precisión de microsegundos.

Requisitos de los precios

- El sistema deberá poder identificar cada precio de un determinado producto en un día específico.
- El sistema deberá mantener la fecha en la que se obtuvo cada precio para llevar a cabo un historial de precios.
- El sistema deberá mantener el día en el que se ha introducido cada producto.
- El sistema deberá mantener el precio de cada producto en un instante de tiempo en un formato numérico para que se puedan hacer operaciones matemáicas sobre el.

Entidades

- **USUARIO** Esta entidad modela un usuario del sistema y va a contener toda la información propia del usuario.
- **BUSQUEDA**: Esta entidad modela una búsqueda que se haya hecho en el sistema sin tener en cuenta quien la ha hecho.
- **PRODUCTO**: Esta entidad modela un producto que se ha obtenido sin tener en cuenta su precio, solo contiene la informaión que describe el producto.
- **PRECIO_PRODUCTO**: Esta entidad modela el precio de un determinado producto en un instante de tiempo sin incorporar información que describa al producto.
- **ESTABLECIMIENTO**: Esta entidad modela los distintos establecimientos de los que se obtienen los productos.

Relaciones y cardinalidad

Entidad	Cardinalidad	Relación	Entidad	Cardinalidad
USUARIO	1..*	realiza	BUSQUEDA	0..*
BUSQUEDA	1..*	contiene	PRODUCTO	0..*
PRODUCTO	0..*	pertenece	ESTABLECIMIENTO	1..1
PRODUCTO	1..*	tiene	PRECIO_PRODUCTO	1..*

- Como mínimo, ¿cuántas **búsquedas** puede realizar un **usuario**? El mínimo de búsquedas que puede hacer un usuario es ninguna.
- Como máximo, ¿cuántas **búsquedas** puede realizar un **usuario**? Un usuario puede hacer como máximo varias búsquedas.
- Como mínimo, ¿cuántos **usuarios** pueden haber hecho la misma **búsqueda**? La misma búsqueda se puede haber hecho como mínimo por un usuario porque sino no existiría esa búsqueda.
- Como máximo, ¿cuántos **usuarios** pueden haberhecho la misma **búsqueda**? La misma búsqueda puede hacerse como máximo por varios usuarios.
- Como mínimo, ¿cuántos **productos** puede contener una **búsqueda**? El mínimo de productos que puede contener una búsqueda es ninguno.
- Como máximo, ¿cuántos **productos** puede contener una **búsqueda**? El máximo número de productos que puede contener una búsqueda es varios productos.
- Como mínimo, ¿a cuántas **búsquedas** puede estar asociado un **producto**? Un producto tiene que estar asociado por lo menos a una búsqueda.
- Como máximo, ¿a cuántas **búsquedas** puede estar asociado un **producto**? Un producto puede estar asociado como máximo a varias búsquedas.

- Como mínimo, ¿a cuántos **establecimientos** puede pertenecer un **producto**? Un producto como mínimo puede pertenecer a un establecimiento.
- Como máximo, ¿a cuántos **establecimientos** puede pertenecer un **producto**? Un producto como máximo puede pertenecer a un establecimiento.
- Como mínimo, ¿cuántos **productos** pueden pertenecer a un **establecimiento**? Como mínimo puede no haber ningún producto asociado a un establecimiento.
- Como máximo, ¿cuántos **productos** pueden pertenecer a un **establecimiento**? Cómo máximo puede haber varios productos asociados a un establecimiento.
- Como mínimo, ¿cuántos **precios** puede tener un **producto**? Como mínimo un producto puede tener un precio.
- Como máximo, ¿cuántos **precios** puede tener un **producto**? Como máximo un producto puede tener varios precios.
- Como mínimo, ¿cuántos **productos** pueden estar asociados al mismo **precio**? Como mínimo un producto tiene que estar asociado a un precio.
- Como máximo, ¿cuántos **productos** pueden estar asociados al mismo **precio**? Como máximo un producto puede estar asociado a varios precios.

Identificadores de las entidades

- **USUARIO**
  - COD_USUARIO: Código del usuario dentro del sistema, este atributo en ningún momento está a la vista del usuario y solo tiene utilidad dentro del sistema para poder identificar al usuario inequivocamente.
- **BUSQUEDA**
  - COD_BUSQUEDA: Código de la búsqueda.
- **PRODUCTO**
  - COD_PRODUCTO: Código del producto dentro de la fuente de datos.
  - COD_ESTABLECIMIENTO: Código de la fuente de datos donde se encontró el producto, este atributo hace referencia a una columna de la tabla ESTABLECIMIENTO.
- **PRECIO_PRODUCTO**
  - COD_PRODUCTO: Código del producto dentro de la fuente de datos.
  - COD_ESTABLECIMIENTO: Código de la fuente de datos donde se encontró el producto, este atributo hace referencia a una columna de la tabla ESTABLECIMIENTO.
  - DIA_PRECIO_PRODUCTO: Dia en la que se ha obtenido ese precio.
- **ESTABLECIMIENTO**
  - COD_ESTABLECIMIENTO: Código de la fuente de datos dentro del sistema.

Identificadores de las relaciones

- A la relación que une la tabla USUARIO con tabla BUSQUEDA la voy a llamar USUARIO_BUSQUEDA, esta relación va a representar las distintas búsquedas que hace un usuario a lo largo del tiempo y contiene los siguientes atributos:
  - COD_USUARIO: Código del usuario que ha realizado la búsqueda.
  - COD_BUSQUEDA: Código de la búsqueda que ha realizado.
  - FX_BUSQUEDA: Fecha en la que se realizó la búsqueda.

  Como se puede ver esta es una entidad débil ya que su existencia depende de dos entidades fuertes (depende de la entidad USUARIO y de la entidad BUSQUEDA). Aparte de los atributos que obtiene de las entidades fuertes de las que depende he añadido el discriminante de la FX_BUSQUEDA para el caso de que un usuario haga la misma búsqueda varias veces.

- A la relación que une las la tabla BUSQUEDA con la tabla PRODUCTOS la voy a llamar RESULTADOS, esta relación representa a qué búsqueda pertenece cada producto ya que un mismo producto puede pertenecer a varias búsquedas. Contiene los siguientes atributos:
  - COD_BUSQUEDA: Código de la búsqueda que produjo los resultados.
  - COD_PRODUCTO: Código del producto que se encontró al hacer esa búsqueda.
  - COD_ESTABLECIMIENTO: Código del establecimiento donde se encontró el producto.

  Esta también es una entidad débil ya que se forma con los identificadores de las entidades BUSQUEDA y la tabla PRODUCTOS.

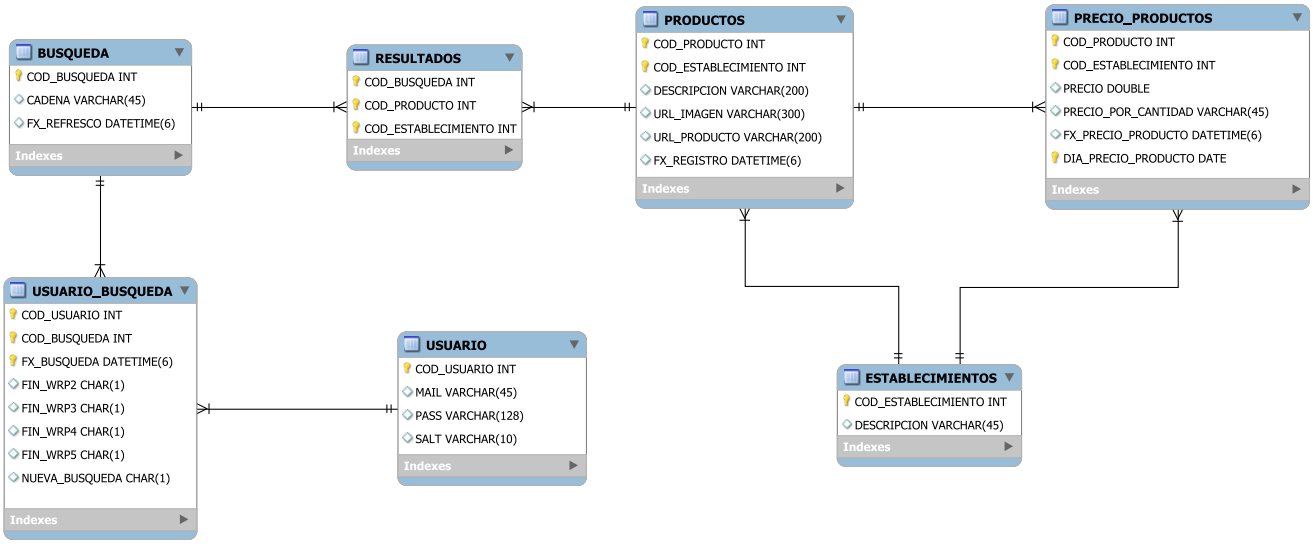

Figura 40: Esquema de la BBDD

Esquema de la base de datos

TABLAS

USUARIO

1. **Atributos**

   - COD_USUARIO: Código numérico del usuario en la base de datos. Este código tiene la propiedad de ser autoincrementado por lo que no hay que ocuparse de el a la hora de crear usuarios.
   - MAIL: Cadena de texto con la que inicia sesión el usuario. A pesar de que en la base de datos no esté reflejado ninguna restricción con respecto a su formato esta cadena de texto va a tener un formato de mail y se va a validar dicho formato en capas superiores.
   - PASS: Hash de la contraseña del usuario.
   - SALT: Cadena que se le añade a la contraseña para generar el hash y hacerla más resistente a ataques por fuerza bruta.

2. **Clave primaria**

   - COD_USUARIO: Se ha elegido este atributo como clave primaria ya que al tener la propiedad de ser autoincrementado no puede repetirse gracias a que el sistema gestor de base de datos se encarga de que cada nueva inserción tenga un valor nuevo.

USUARIO_BUSQUEDA

1. **Atributos**

   - COD_USUARIO: Código numérico que hace referencia al atributo COD_USUARIO de la tabla USUARIO y refleja qué usuario ha hecho esa búsqueda.
   - COD_BUSQUEDA: Código numérico que hace refencia al atributo COD_BUSQUEDA de la tabla BUSQUEDA y refleja qué búsqueda acaba de hacer el usuario.
   - FX_BUSQUEDA: Momento en el que realizó la búsqueda ese usuario.
   - FIN_WRP2: Cadena de texto que permite representar cuando ha terminado el wrapper 2.
   - FIN_WRP3: Cadena de texto que permite representar cuando ha terminado el wrapper 3.
   - FIN_WRP4: Cadena de texto que permite representar cuando ha terminado el wrapper 4.
   - FIN_WRP5: Cadena de texto que permite representar cuando ha terminado el wrapper 5.
   - NUEVA_BUSQUEDA: Cadena de texto que permite representar qué tipo de búsqueda ha sido, si ha sido una búsqueda nueva (aquella en la que se tiene que ir a las fuentes de datos a por la información) o una búsqueda no nueva (aquella búsqueda en la que se va a la base de datos a por la información).

2. **Clave primaria**

   - COD_USUARIO
   - COD_BUSQUEDA
   - FX_BUSQUEDA

   Esta combinación me permite poder registrar la misma búsqueda de un usuario varias veces cuando se haya producido en instantes de tiempo diferentes.

La relación entre las tabla **USUARIO** y **USUARIO_BUSQUEDA** permite que distintos usuarios hagan referencia a la misma búsqueda con lo que se evitan duplicidades.

BUSQUEDA

1. **Atributos**

   - COD_BUSQUEDA: Código de la búsqueda. Este código tiene la propiedad de ser autoincrementado por lo que no hay que ocuparse de el a la hora de crear búsquedas.
   - CADENA: Cadena sobre la que se hizo la búsqueda.
   - FX_REFRESCO: Última fecha en la que se realizó la búsqueda.

2. **Clave primaria**

   - COD_BUSQUEDA: Se ha elegido este atributo como clave primaria ya que al tener la propiedad de ser autoincrementado no puede repetirse gracias a que el sistema gestor de base de datos se encarga de que cada nueva insercción tenga un valor nuevo.

   Aunque no forme parte de la clave primaria el atributo CADENA tiene la propiedad UNIQUE para que no se repita.

PRODUCTOS

1. **Atributos**

   - COD_PRODUCTO: Código del producto dentro de la fuente de datos donde se ha obtenido.
   - COD_ESTABLECIMIENTO: Código del establecimiento al que pertenece.
   - DESCRIPCION: Breve descripción del producto, esta descripción se obtiene de la fuente de datos.
   - URL_IMAGEN: Dirección web a una imagen del producto.
   - URL_PRODUCTO: Dirección web a la página donde se encuentra el producto.
   - FX_REGISTRO: Última fecha en la que se obtuvo el producto. Almacenando la última fecha en la que se obtuvo cada producto puedo saber cuales son los productos que están disponibles actualmente y qué productos ya no lo están.

2. **Clave primaria**

   - COD_PRODUCTO
   - COD_ESTABLECIMIENTO

   Se ha decidido tomar estos dos atributos en vez de tomar simplemente el COD_PRODUCTO que se obtiene de cada fuente de datos porque podría darse el caso en que dos productos de dos fuentes de datos tuvieran el mismo código y solo pudiera almacenar un producto en la base de datos. Con este sistema consigo que aunque se dé este caso pueda almacenar ambos productos.

RESULTADOS

1. **Atributos**

   - COD_BUSQUEDA: Código de la búsqueda a la que pertenece ese producto. Este código hace referencia al atributo COD_BUSQUEDA de la tabla BUSQUEDA.
   - COD_PRODUCTO: Código del producto asociado a esa búsqueda que se encontró. Este código hace referencia al atributo COD_PRODUCTO de la tabla PRODUCTOS.
   - COD_ESTABLECIMIENTO: Código del establecimiento donde estaba el producto que se encontró. Este código hace referencia al atributo COD_ESTABLECIMIENTO de la tabla PRODUCTOS.

2. **Clave primaria**

   - COD_BUSQUEDA
   - COD_PRODUCTO
   - COD_ESTABLECIMIENTO

   Esta combinación de atributos me permite registrar qué productos pertenecen a una determinada búsqueda incluso cuando un producto pertenece a varias búsquedas.

PRECIO_PRODUCTOS

1. **Atributos**

   - COD_PRODUCTO: Código del producto. Este código hace referencia al atributo COD_BUSQUEDA de la tabla BUSQUEDA.
   - COD_ESTABLECIMIENTO: Código del establecimiento al que pertenece. Este código hace referencia al atributo COD_ESTABLECIMIENTO de la tabla ESTABLECIMIENTOS.
   - PRECIO: Precio que tiene el producto en ese instante de tiempo.
   - PRECIO_POR_CANTIDAD: Precio del producto por unidad de medida.

- FX_PRECIO_PRODUCTO: Fecha en la que se obtuvo ese precio.
- DIA_PRECIO_PRODUCTO: Día en el que se obtuvo el precio.

2. **Clave primaria**

   - COD_PRODUCTO
   - COD_ESTABLECIMIENTO
   - DIA_PRECIO_PRODUCTO

   Esta combinación de atributos me permite registrar un solo precio para un producto al día.

ESTABLECIMIENTOS

1. **Atributos**

   - COD_ESTABLECIMIENTO: Código numérico del establecimiento. Este código no tiene ningún sentido fuera de la aplicación.
   - DESCRIPCION: Breve descripción del establecimiento.

2. **Clave primaria**

   - COD_ESTABLECIMIENTO

Motor

A pesar de que existen varios motores de almacenamiento en MySQL a la hora de elegir uno mi decisión final estaba entre **InnoDB** y **MyISAM**.

- Por un lado **InnoDB** tiene soporte de transacciones (conforme a ACID) con capacidad de commit y rollback y también soporta restricciones de integridad referencial. [21]
- Por otro lado **MyISAM** no soporta la integridad referencial ni es transaccional. Estas características anteriores llevan a que este motor sea bastante más rápido que InnoDB.

Como desde un primer momento tenía claro que la BBDD tenía que tener restricciones fuertes, soporte para transacciones, rollback y las insercciones iban a ser muy frencuentes decidí utilizar el motor InnoDB

4.3.5. Comunicación del controlador con el navegador web cliente

Paso de información del controlador al navegador

La comunicación del navegador con el servidor a la hora de obtener los productos se realiza mediante llamadas Ajax mientras que la forma que se ha decidido utilizar para pasar la información es mediante JSON [22]. Algunas de las ventajas al utilizar este sistema son:

- Los objetos JSON se autodescriben y son fáciles de entender.
- Es independiente del lenguaje.
- Existen plugins para JavaScript que permiten parsearlo fácilmente.
- Existen bibliotecas para Java que permiten convertir un objeto en un JSON de forma transparente.
- JSON necesita menos tags con lo que se reduce su tamaño respecto a otra opción como pudiera ser XML.
- Es más sencillo de leer.

Cuando tengo que devolver la información en el servidor al cliente podría haber creado el objeto JSON manualmente pero decidí buscar una biblioteca que hiciera ese trabajo y encontré Gson [23], es una librería para Java que convierte objetos Java en su representación para JSON. También se podría utilizar para convertir un objeto JSON en su equivalente objeto Java aunque esto no se utiliza en la aplicación:

```
Gson gson = new GsonBuilder().excludeFieldsWithoutExposeAnnotation().create();
String json = gson.toJson(salida);
```

En este ejemplo se convierte un objeto Java en su representación JSON con la biblioteca Gson. Una vez que se ha recibido la información en el navegador se podría haber optado por parsear ese objeto JSON manualmente pero se ha decidido utilizar una biblioteca para JavaScript que hace esta tarea, esa biblioteca es Jquery [28]. El método de esta biblioteca que he usado para esta tarea es "parseJSON()" que se encarga de tomar un string JSON y devolver el objeto JavaScript resultante, una vez que tengo ese objeto JavaScript puedo recorrerlo e imprimirlo en el JSP para que el usuario pueda verlo. Un ejemplo del objeto JSON que se transmite desde el navegador al cliente es el siguiente:

```
["E",
"9",
"",
0,
"",
[
{
"codProducto":"10156",
"codEstablecimiento":"3",
"nombreEstablecimiento":"LUPA",
"descripcion":"AGUA VALTORRE MINERAL 33 CL.",
```

```
"urlImagen":"http://virtual.lab.inf.uva.es:20052/TFG_WEB/images
/img_noDisponible.gif",
"urlProducto":"http://www.lupaonline.com/lupa391
/agua-valtorre-mineral-33-cl.html",
"precio":0.15,
"precioPorCantidad":"El Litro sale a: 0,45 euro"
},
{"codProducto":"8010096",
"codEstablecimiento":"4",
"nombreEstablecimiento":"GADIS",
"descripcion":"AGUA S/G UNIAQUA BTLLA. PET 33CL.",
"urlImagen":"http://localhost:8080/TFG_WEB/images
/img_noDisponible.gif",
"urlProducto":"http://www.gadisline.com/web
/verProducto.jsp?\u0026productoId\u003d8010096",
"precio":0.16,
"precioPorCantidad":"0,48 euro/EL LITRO"}
],
"2014-08-31 17:56:37"]
```


Como se puede ver este JSON tiene 7 partes:

1. Tipo de búsqueda: Con este parámetro especifico si la búsqueda existía ("E") o no existía ("N").
2. Código de búsqueda: Código de la búsqueda que ha creado el usuario.
3. Fecha mayor: Este parámetro se utiliza cuando la búsqueda no existía y se utiliza para saber cual es la fecha de obtención del último producto que se ha enviado.
4. Número de búsqueda: Este parámetro se utiliza cuando la búsqueda si que existía y sirve para delimitar el númeo de productos cuando se obtienen.
5. Código del error: Este parámetro viene con un valor si no había productos para esa búsqueda ("NP"), no había más productos para esa búsqueda ("NMP") o se produce un error de cualquier tipo en Java ("ERROR").
6. Productos: Es la lista con los productos.
7. Fecha de creación de la búsqueda: Fecha en la que se creó la búsqueda para ese usuario.

Evitar doble submit

Uno de los problemas que me encontré en esta parte es la de evitar que un usuario pueda hacer doble submit del formulario de búsqueda. A pesar de que la aplicación por debajo esté preparada para soportar este supuesto mediante tratamiento de excepciones y creación de claves primarias en la base de datos es un problema que debería evitarse en este punto, en el navegador del cliente. Para evitar este problema hago uso de dos funcionalidades de JQuery:

- Esta biblioteca permite asociar datos a un elemento HTML de modo que he añadido al formulario una variable para determinar si ya se ha hecho submit de ese formulario y así antes de hacer la llamada Ajax que hace la búsqueda compruebo si esa variable es igual a true (se ha hecho submit) o a false (no se ha hecho submit). Como se tiene que permitir que un usuario haga todas las búsquedas que quiera el único espacio de tiempo en el que está bloqueado el submit de ese formulario es mientras se está haciendo una búsqueda.

- A pesar de que el sistema anterior debería funcionar, si un usuario impaciente hace dos veces click en el botón de buscar es probable que se produzca un error por tanto una vez que se ha hecho submit del formulario y se hace la llamada Ajax espero un segundo hasta volver a permitir que se vuelva a hacer submit del formulario.

4.4. Aspectos de la BBDD

Creación de índices

Definición y tipos A la hora de crear el diseño de una base de datos para una aplicación es necesario crear índices sobre los atributos de las tablas para obtener un mayor rendimiento. Crear índices no es algo que se deba hacer a la ligera ya que aunque es cierto que optimizan las consultas también pueden penalizar las insercciones. Para crear índices hay que tener en cuenta las consultas que se van a realizar sobre la base de datos para ver sobre qué campos sería necesario hacerlos. Los índices se utilizan para encontrar filas con un valor en una columna específico. Sin los índices MySQL tiene que leer todas las filas de la tabla para encontrar el resultado. Si la tabla tiene un índice para una columna MySQL puede rapidamenete determinar la posición en la que buscar en vez de tener que recorrer toda la tabla, si la tabla tiene 1000 filas esto es por lo menos 100 veces más rápido [30]. MySQL tiene los siguientes índices:

- **PRIMARY KEY**: Este índice debe ser único y no se admite el almacenamiento de NULL.

- **INDEX**: Permite crear índices sobre una columna, varias columnas o sobre partes de una columna.

- **UNIQUE**: Este tipo de índices no permite el almacenamiento de valores iguales.

- **FULLTEXT**: Permite realiza búsquedas de palabras, solo puede usarse sobre columnas CHAR, VARCHAR o TEXT y en tablas con motor MyISAM.

Índices en la aplicación

- **Tabla USUARIO**: Las consultas que se hacen sobre esta tabla se hacen sobre los campos MAIL y PASS, lo ideal hubiera sido que se pudiera haber creado un índice de tipo FULLTEXT sobre estos campos pero el motor InnoDB no permite estos tipos de índices. Existe un índice de tipo PRIMARY que afecta a la clave primaria y luego un índice de tipo UNIQUE sobre el campo MAIL para que no pueda repetirse y también sirve como índice de tipo UNIQUE.

- **Tabla USUARIO_BUSQUEDA**: En esta tabla se hacen consultas sobre el campo COD_USUARI este campo al formar parte de la clave primaria tiene asignado un índice de tipo PRIMARY KEY.
- **Tabla BUSQUEDA**: Las búsquedas son sobre la tabla CADENA y COD_BUSQUEDA. Por un lado la columna CADENA tiene un índice de tipo UNIQUE que por un lado evita que se inserte varias veces la misma cadena y por otro lado que haya un índice en ese campo. La columna COD_BUSQUEDA al formar parte de la clave primaria tiene un índice de tipo PRIMARY KEY.
- **Tabla PRODUCTOS**: Se utilizan los campos COD_PRODUCTO y COD_ESTABLECIMIENTO que al formar parte de la clave primaria tiene un índice de tipo PRIMARY KEY.
- **Tabla RESULTADOS**: Las búsquedas realizadas son sobre los campos COD_BUSQUEDA que al formar parte de la clave primaria tiene un índice de tipo PRIMARY KEY.
- **Tabla PRECIO_PRODUCTOS**: Se utilizan los campos COD_PRODUCTO, COD_ESTABLEC y DIA_PRECIO_PRODUCTO. Al ser los 3 campos parte de la clave primaria tiene un índice de tipo PRIMARY KEY.
- **Tabla ESTABLECIMIENTOS**: Solo se utiliza el campo COD_ESTABLECIMIENTO y al formar parte de la clave primaria tiene asignado un índice de tipo PRIMARY KEY.

Vistas

¿Qué son? Las vistas en MySQL son queries almacenadas que cuando son invocadas producen un conjunto de resultados por tanto se puede decir que una vista actua como una tabla virtual.

Vistas en la aplicación En esta aplicación he utilizado las vistas para simplificar el acceso al catálogo de productos y de precio de productos:

```
CREATE VIEW VISTA_BUSQUEDA AS
SELECT b.COD_BUSQUEDA, p.COD_PRODUCTO,p.FX_REGISTRO,pre.PRECIO,
p.DESCRIPCION, p.COD_ESTABLECIMIENTO,
pre.PRECIO_POR_CANTIDAD ,p.URL_IMAGEN, p.URL_PRODUCTO
FROM BUSQUEDA b
INNER JOIN RESULTADOS r
ON b.COD_BUSQUEDA = r.COD_BUSQUEDA
INNER JOIN PRODUCTOS p
ON r.COD_PRODUCTO = p.COD_PRODUCTO
AND r.COD_ESTABLECIMIENTO = p.COD_ESTABLECIMIENTO
INNER JOIN PRECIO_PRODUCTOS pre
ON p.COD_PRODUCTO = pre.COD_PRODUCTO
AND p.COD_ESTABLECIMIENTO = pre.COD_ESTABLECIMIENTO
AND pre.FX_PRECIO_PRODUCTO = (
```

```
SELECT MAX(FX_PRECIO_PRODUCTO)
FROM PRECIO_PRODUCTOS
WHERE COD_PRODUCTO = pre.COD_PRODUCTO
AND COD_ESTABLECIMIENTO = pre.COD_ESTABLECIMIENTO)
AND p.FX_REGISTRO >= b.FX_REFRESCO;
```

Gracias al uso de esta vista puedo referenciar los productos de una forma tan fácil como esta:

- Query usada para seleccionar los productos por precio

```
SELECT *
FROM VISTA_BUSQUEDA
WHERE COD_BUSQUEDA = ?
ORDER BY PRECIO LIMIT ?,?;
```

- Query usada para seleccionar los productos en función de la fecha en la que se obtuvieron

```
SELECT *
FROM VISTA_BUSQUEDA
WHERE COD_BUSQUEDA = ?
AND FX_REGISTRO > ?
ORDER BY FX_REGISTRO LIMIT ?;
```

Procedimientos almacenados

¿Qué son Un procedimiento almacenado en MySQL es un conjunto de comandos SQL que pueden almacenarse en el servidor. Estos procedimientos pueden ser especialmente útiles cuando hay varias aplicaciones cliente que necesitan hacer uso de la misma operacion en la base de datos. Otro ámbito en el que son muy importantes es en el de la seguridad ya que los clientes hacen referencia a los procedimientos almacenados en vez de tratar con las tablas.

Los procedimientos almacenados pueden conllevar una mejora en el rendimiento de aquellas operaciones que necesiten tratar con los datos almacenados. De esta forma los datos en los procedimientos almacenados se encuentran en la misma máquina mientras que de otra forma, en una aplicación cliente por ejemplo, los datos tendrían que pasarse del servidor de base de datos al cliente y allí realizarse las operaciones.

Procedimientos en la aplicación Como se ha explicado en el punto del diseño de la solución, se podría establecer como política de la aplicación el eliminar aquellas búsquedas que no hayan generado resultados para evitar que diariamente se intenten recargar los datos de esas búsquedas. Incluso se podría hacer que se eliminaran determinadas búsquedas no vacías dependiendo de algún criterio estadístico. Por este motivo he creado en la base de datos un procedimiento almacenado llamado "BORRAR_BUSQUEDAS_VACIAS" que acepta como argumento un número entero y elimina de la base de datos aquellas búsquedas que han generado ese número de resultados de modo que se podría eliminar las búsquedas vacías con este procedimiento. El procedimiento almacenado tiene el siguiente código:

```
CREATE DEFINER=‘root‘@‘%‘ PROCEDURE ‘BORRAR_BUSQUEDAS_VACIAS‘(IN numProductos INT)
BEGIN

DECLARE done INT DEFAULT FALSE;
DECLARE codBusqueda INT;
DECLARE cur CURSOR FOR SELECT COD_BUSQUEDA FROM TFG.RESULTADOS
GROUP BY COD_BUSQUEDA HAVING COUNT(COD_PRODUCTO) = numProductos;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

start_loop: LOOP
FETCH cur into codBusqueda;
IF done THEN
LEAVE start_loop;
END IF;

-- Borro la tabla TFG.RESULTADOS
DELETE FROM TFG.RESULTADOS WHERE COD_BUSQUEDA = codBusqueda;
-- Borro la tabla TFG.USUARIO_BUSQUEDA
DELETE FROM TFG.USUARIO_BUSQUEDA WHERE COD_BUSQUEDA = codBusqueda;
-- Borro la tabla TFG.BUSQUEDA
DELETE FROM TFG.BUSQUEDA WHERE COD_BUSQUEDA = codBusqueda;

end LOOP;

END
```

Como se puede ver este procedimiento es muy sencillo y hace uso de un cursor para recuperar todas las búsquedas que han generado un determinado número de resultados. El paso siguiente es, por cada búsqueda, eliminar toda su información de las tablas en las que está presente.

Consultas permitidas

Consulta del historial de búsquedas de un usuario

```
SELECT ub.COD_BUSQUEDA, ub.FX_BUSQUEDA
FROM USUARIO_BUSQUEDA ub
INNER JOIN USUARIO u
ON u.LOGGIN = ?
AND u.COD_USUARIO = ub.COD_USUARIO;
```

Consulta del historial de precios de un producto

```
SELECT *
```

```
FROM PRECIO_PRODUCTOS WHERE COD_PRODUCTO = ?
AND COD_ESTABLECIMIENTO = ?
ORDER BY FX_PRECIO_PRODUCTO;
```

Consulta de los productos disponibles asociados a una cadena

```
SELECT *
FROM BUSQUEDA b
INNER JOIN RESULTADOS r
ON b.COD_BUSQUEDA = r.COD_BUSQUEDA
INNER JOIN PRODUCTOS p
ON p.COD_PRODUCTO = r.COD_PRODUCTO
AND p.COD_ESTABLECIMIENTO = r.COD_ESTABLECIMIENTO
AND p.FX_REGISTRO >= b.FX_REFRESCO
AND b.CADENA = ?;
```

Consulta de los productos disponibles asociados a una búsqueda El poder hacer esta búsqueda es más importante de lo que parece ya que si se van obteniendo productos es necesario poder saber qué produtos se han obtenido a partir de la última búsqueda. Para hacer esta distinción tengo en cuenta la fecha en la que se ha hecho la búsqueda y la última fecha en la que se han obtenido los productos.

```
SELECT p.COD_PRODUCTO, p.COD_ESTABLECIMIENTO
FROM BUSQUEDA b
INNER JOIN RESULTADOS r
ON b.COD_BUSQUEDA = r.COD_BUSQUEDA
AND b.COD_BUSQUEDA = ?
INNER JOIN PRODUCTOS p
ON r.COD_PRODUCTO = p.COD_PRODUCTO
AND r.COD_ESTABLECIMIENTO = p.COD_ESTABLECIMIENTO
AND p.FX_REGISTRO > b.FX_REFRESCO;
```

Consultas no permitidas

Consulta del historial de precios de una búsqueda pasada

Consulta del historial de productos generados por una búsqueda pasada

4.5. Copias de seguridad

Como el valor de esta aplicación son los datos que almacena considero una parte clave el llevar a cabo algún mecanismo de copias de seguridad. En este caso ya que estamos utilizando MySQL disponemos de la herramienta mysqldump [36]. Este programa se puede usar para hacer copias de seguridad de una base de datos o de una colección de bases de datos y exportar su contenido a comandos SQL para crear las tablas y rellenarlas. Algunas de las opciones que permite para crear una copia de una base de datos (o de alguna tabla de una base de datos) son:

- Copia de seguridad de una única base de datos

  ```
  mysqldump -u[USER] -p[PASSWORD] [DB_NAME] > salida.sql
  ```

- Copia de seguridad de varias bases de datos

  ```
  mysqldump -u[USER] -p[PASSWORD] --databases [DB_NAME1] [DB_NAME1] > salida.sql
  ```

- Copia de seguridad de **todas** las bases de datos

  ```
  mysqldump -u[USER] -p[PASSWORD] --all-databases > salida.sql
  ```

- Copia de seguridad de una tabla específica

  ```
  mysqldump -u[USER] -p[PASSWORD] [DB_NAME] [TABLE_NAME] > salida.sql
  ```

Una vez hecha la copia de seguridad de las tablas que se desee para hacer una recupeación bastaría con hacer:

```
mysql -u[USER] -p[PASSWORD] [DB_NAME] < salida.sql
```

Una opción muy interesante de este comando es la de hacer una copia de seguridad de los procedimientos almacenados que hay en la base de datos, para ello se ejecutaría el siguiente comando:

```
mysqldump -u[USER] -p[PASSWORD] --routines [DB_NAME] >> salida.sql
```

Aunque de menor importante, considero que los ficheros de logs de la aplicación también son muy importantes en todo momento y creo que deberían añadirse al conjunto de ficheros del que se debe hacer copia de seguridad por eso aparte de la base de datos hago una copia de seguridad de los ficheros de logs.

Políticas de copias de seguridad

A la hora de decidir la política de copias de seguridad en la aplicación se consideraron las siguientes opciones:

- **Copia de seguridad completa**

  Como su nombre indica, esta copia de seguridad crea una copia de toda la información que se quiera respaldar. Este método resulta muy rápido a la hora de restaurar pero el gran inconveniente es que se crea información duplicada que se respalda. Este método es el indicado cuando la cantidad de datos no es muy grande y/o cuando se necesita la capacidad de restaurar la información muy rapidamente.

- **Copia de seguridad incremental**

Este mecanismo proporciona una copia de seguridad de los datos que han cambiado desde el último backup, es decir, si se crea una copia de seguridad incremental un jueves solo se copian los datos que se han creado desde la última copia de seguridad. Este método es mucho más rápido que la copia de seguridad completa ya que se hace una copia de menos datos pero en contrapartida el tiempo de restauración de las copias de seguridad es mayor ya que la copia de seguridad debe reconstruyrse desde la última copia de seguridad completa y con todas las copia de seguridad incrementales realzada desde entonces. Esta política de copias de seguridad considero que es la adecuada cuando se tratan grandes volúmenes de información y no resulta práctica copiar toda la información y se puede asumir un tiempo "alto" de restauración.

- **Copia de seguridad diferencial**

  Mediante las copias de seguridad diferenciales se crea una copia de los datos que se han modificado desde la última copia de seguridad completa. Como se puede ver este mecanismo es un punto intermedio entre las copias de seguridad completas y las copia de seguridad incrementales ya que realiza un mecanismo similar al de las copias de seguridad incrementales creando copias parciales pero el tiempo de restauración es menor que en el caso de las copias de seguridad incrementales.

A la hora de decidirme entre el sistema de copia de seguridad di prioridad a dos puntos. Primero hay que considerar que el volumen de datos a pesar de no ser pequeño no es muy alto y es asumible el crear una copia de seguridad total. Por otro lado se quiere crear un mecanismo mediante el que en caso de error se pueda recuperar el estado de la aplicación en un menor tiempo. Por estos dos puntos se ha decidido utilizar una política de seguridad completa.

4.6. Seguridad de los datos

Uno de los puntos que se han tenido en cuenta a la hora de realizar la aplicación es la de garantizar la seguridad de los datos. En la aplicación se cifran las contraseñas de los usuarios para que en el caso de que alguien pudiera tener acceso a la base de datos no pudiera obtenerlas. La decisión que se ha tomado es la de almacenar el hash de las contraseñas en la base de datos y no disponer de ningún sistema mediante el que se pueda recuperar la contraseña original. Este sistema tiene el claro beneficio de que es muy difícil obtener la contraseña original a partir del hash pero la parte negativa es que si un usuario se olvida de su contraseña no podría acceder a la aplicación ni recuperar la contraseña. A pesar de que no se haya implementado ningún mecanismo de recuperación de contraseña, no sería difícil elaborar un sistema mediante el que se creara una nueva contraseña para el usuario y se le enviara por correo para que pudiera volver a tener acceso a la aplicación.

Para obtener el hash criptográfico de la contraseña he decidido utilizar el algoritmo SHA-512, este es un algoritmo hash de un solo sentido. Voy a utilizar este algoritmo junto a un salt. Un salt es una cadena de texto aleatoria que se utiliza junto a la contraseña para formar el hash y

así prevenir los ataques por diccionario, esta cadena se guarda junto al hash en base de datos y se utiliza para validar la contraseña [37]. El procedimiento para almacenar una contraseña es:

1. Generar un salt para ese usuario.
2. Unir la contraseña y el salt para generar el hash con el algoritmo SHA-512.
3. Almacenar el hash y el salt en la base de datos para futuras validaciones de la contraseña.

Para validar una contraseña el procedimiento es:

1. Recuperar el salt y el hash de ese usuario de la base de datos.
2. Unir el salt y el password introducido por el usuario y calcular el hash.
3. Comparar ese hash generado con el que estaba almacenado en base de datos.

Para obtener tanto el hash como el salt he utilizado el API java.security [38]:

5. Conclusiones

Una de las conclusiones más importantes que he sacado a la hora de crear este sistema es que conocer el esquema de una fuente de datos que no es propia es muy difícil. El no disponer de ningún diagrama de los datos con los que estoy tratando o conocer el formato de esos datos fuerza a tener que aprender cómo funciona la página. La mayoría de las veces no vale solamente con aprender cómo funciona un sistema porque el sistema va evolucionando sin que tu tengas información de cómo cambia.

Otro punto del que me he dado cuenta es que por muy bien que hagas un sistema de este tipo, es decir, un sistema que depende de otras aplicaciones, estas insertando una incertidumbre respecto a la fiabilidad de tu sistema. Esto viene de que no puedes asegurar la disponibilidad de sistemas que no conoces y tampoco puedes asegurar la obtención de unos datos de los que no conoces su formato. Por tanto creo que es conveniente el conocer a la perfección (en la medida de lo posible) los esquemas de las fuentes de datos de las que se obtiene la información. A pesar de haber llegado a conocer las fuentes de datos pueden darse casos extraordinarios debido a diferentes políticas en la empresa que puedan llevar a situaciones de error o a situaciones en las que no funcione correctamente la aplicación.

A la hora de diseñar los elementos que se van a ocupar de comunicarse con las fuente de datos hay que tener en cuenta varios aspectos. Por un lado hay que crearlos de la manera más simple posible para facilitar su modificación en caso de que haya que cambiarlos. Estos cambios son muy probables ya que el web scrapping es muy dependiente del esquema de una página web por lo que un cambio mínimo puede hacer que deje de funcionar ese wrapper. A la sencillez del cambio de ese wrapper hay que sumarle su eficiencia, estos componentes tienen que ser todo lo rápidos que lo permita el webdriver con el que se esté tratando evitando la obtención innecesaria de páginas e incluso reduciendo el timeout a la hora de obtener esas páginas.

Cuando se utilizan fuentes de datos de terceros hay que intentar ser respetuosos con esos sitios web. Con esto quiero decir que aunque esos sitios deberían permitir varias peticiones simultaneas no conviene sobrecargar esos sistemas, ya no solo porque si se sobrecargan podrían llegar a estar fuera de servicio sino porque se podría incurrir en actividades ilegales.

Por último, cuando se utiliza una arquitectura basada en data warehouse hay que desarrollar algún sistema por el cual se refresque esa información. La creación de este sistema no es trivial y hay que garantizar que se consigan los objetivos con un coste mínimo (a no ser que se esté dispuesto a asumir un coste mayor por alguna política que se haya tomado).

6. Trabajo futuro

A pesar de que la aplicación está terminada para su uso por un usuario final, se podría dotar a la aplicación de la siguiente funcionalidad:

Recuperación de contraseñas Ahora mismo la aplicación no cuenta con ningún sistema de recuperación de contraseñas en el caso de que el usuario la pierda. Como lo que se almacena en la base de datos es el hash de la contraseña y ningún mecanismo de recuperación sería interesante mediante algún mecanismo generar una contraseña nueva para el usuario y enviarsela por mail.

Historíal de búsquedas de un usuario Como la aplicación registra todas las búsquedas que realiza un usuario se le podría mostrar un listado con todas las búsquedas que ha hecho y por ejemplo el precio más bajo de cada búsqueda.

Estadísticas en las búsquedas de un usuario Como he mencionado en el punto anterior, se guardan todas las búsquedas y podrían mostrarse el número total de búsquedas del usuario, media de búsqueda diaria, búsquedas mensuales,etc.

Filtro en función del precio A la hora de realizar una búsqueda contra las fuentes de datos no sería difícil filtrar los resultados que se esperan obtener en función del precio del producto.

Filtro en función de la fuente de datos Se podría limitar el resultado de productos a un centro comercial específico en vez de a los cuatro directamente.

Transformación del precio por cantidad Ahora mismo se toma de las fuentes de datos el precio total del producto y el precio por cantidad, mediante transformaciónes en la herramienta ETL se podrían hacer las búsquedas por el precio por cantidad en vez de por el precio final.

Aviso de precio de productos Sería interesante para un trabajo futuro el implementar un mecanismo mediante el cual el cliente se subscriba a un producto y le lleguen notificaciones al correo cuando el precio del producto que ha solicitado se encuentre por debajo de un umbral.

7. Localización de la aplicación

La aplicación que se ha descrito en la presente memoria se encuentra incluida en la documentación adjunta a esta memoria. A mayores se encuentra alojada en los servidores de la escuela, para acceder a ella basta con entrar en la dirección `http://virtual.lab.inf.uva.es:20052/TFG`. En esta página está disponible el registro de usuarios, de cualquier modo he creado un usuario para las pruebas sobre la aplicación:

```
loggin: tfg@gmail.com
pass: 123456789
```

Es importante indicar que como uno de los puntos interesantes de la aplicación es el del histórico de precios, es interesante hacer una búsqueda que ya existiera por lo que recomiendo hacer la búsqueda "agua" para que así se pueda ver el registro de precios que mantiene la aplicación.

Referencias

[1] Herbert Schildt. *Java 7: A Beginner's Guide*. McGraw-Hill, Inc., 2010.

[2] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[3] Jérôme Lafosse. *Struts 2: El framework de desarrollo de aplicaciones Java EE*. Ediciones Eni, 2010.

[4] Ian Roughley. *Starting Struts 2*. C4Media, 2007.

[5] Donald Brown, Chad Michael Davis, and Scott Stanlick. *Struts 2 in action*. Dreamtech Press, 2008.

[6] *Struts 2*. `http://struts.apache.org/development/2.x/`, Fecha de acceso: 15-07-2014.

[7] *Interceptors*. `http://struts.apache.org/release/2.3.x/docs/interceptors.html`, Fecha de acceso: 16-07-2014.

[8] *Session management in Struts2*. `http://www.jkstack.com/2013/01/session-management-using-login.html`, Fecha de acceso: 16-07-2014.

[9] *Form Validation Using XML*. `http://struts.apache.org/development/2.x/docs/form-validation-using-xml.html`, Fecha de acceso: 15-07-2014.

[10] *HtmlUnit*. `http://htmlunit.sourceforge.net/`, Fecha de acceso: 16-07-2014.

[11] *JSoup*. `http://jsoup.org/`, Fecha de acceso: 14-07-2014.

[12] *Página supermercado Eroski*. `http://www.compraonline.grupoeroski.com/supermercado/home.jsp`, Fecha de acceso: 14-07-2014.

[13] *Página supermercado Lupa*. `http://www.lupaonline.com/lupa391/`, Fecha de acceso: 14-07-2014.

[14] *Página supermercado Gadis*. `http://www.gadisa.es/gadis/`, Fecha de acceso: 14-07-2014.

[15] *Página supermercado Alcampo*. `http://www.alcampo.es/`, Fecha de acceso: 14-07-2014.

[16] *Core J2EE Patterns - Data Access Object*. `http://www.oracle.com/technetwork/java/dataaccessobject-138824.html`, Fecha de acceso: 13-07-2014.

[17] *c3p0 - JDBC3 Connection and Statement Pooling*. `http://www.mchange.com/projects/c3p0/`, Fecha de acceso: 19-07-2014.

[18] *C3P0 Last Version*. `http://sourceforge.net/projects/c3p0/`, Fecha de acceso: 19-07-2014.

[19] *C3P0 - Configuration Properties*. `http://www.mchange.com/projects/c3p0/#configuration_properties`, Fecha de acceso: 19-08-2014.

[20] *C3P0 - Configuring Connection Testing*. `http://www.mchange.com/projects/c3p0/#configuring_connection_testing`, Fecha de acceso: 19-08-2014.

[21] *MyISAM - Engine.* http://dev.mysql.com/doc/refman/5.0/es/myisam-storage-engine.html, Fecha de acceso: 12-07-2014.

[22] *JSon Tutorial.* http://www.w3schools.com/json/, Fecha de acceso: 25-08-2014.

[23] *google-gson.* https://code.google.com/p/google-gson/, Fecha de acceso: 25-08-2014.

[24] Jake Spurlock. *Bootstrap.* .º'Reilly Media, Inc.", 2013.

[25] *Boostrap tutorial.* http://librosweb.es/bootstrap_3/, Fecha de acceso: 19-08-2014.

[26] Cody Lindley. *jQuery Cookbook: Solutions & Examples for jQuery Developers.* .º'Reilly Media, Inc.", 2009.

[27] Jonathan Chaffer and Karl Swedberg. *Learning jQuery.* Packt Publishing Ltd, 2011.

[28] *jQuery.* http://jquery.com/, Fecha de acceso: 25-08-2014.

[29] Thomas M Connolly and Carolyn E Begg. *Sistemas de Bases de Datos: Un Enfoque Practico Para Diseno, Implementacion y Gestion/Database Systems.* Pearson Education, 2006.

[30] *Índices en MySQL.* http://dev.mysql.com/doc/refman/5.0/es/mysql-indexes.html, Fecha de acceso: 20-07-2014.

[31] *Vistas en MySQL.* http://dev.mysql.com/doc/refman/5.0/es/views.html, Fecha de acceso: 13-08-2014.

[32] *Procedimientos almacenados en MySQL.* http://dev.mysql.com/doc/refman/5.0/es/stored-procedures.html, Fecha de acceso: 13-08-2014.

[33] Steven Nelson and Russell Brown. *Pro data backup and recovery.* Springer, 2011.

[34] *About backup method advantages and disadvantages.* http://www.symantec.com/business/support/index?page=content&id=HOWTO22705, Fecha de acceso: 19-07-2014.

[35] *Descripción de copias de seguridad.* http://support.microsoft.com/kb/136621, Fecha de acceso: 19-07-2014.

[36] *MySQLDump.* http://dev.mysql.com/doc/refman/5.0/es/mysqldump.html, Fecha de acceso: 19-07-2014.

[37] *Algoritmos Hash.* https://crackstation.net/hashing-security.htm, Fecha de acceso: 20-07-2014.

[38] *Generar contraseñas en Java.* http://howtodoinjava.com/2013/07/22/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/, Fecha de acceso: 20-07-2014.

[39] *Log4J2.* http://logging.apache.org/log4j/2.x/, Fecha de acceso: 20-07-2014.

Apéndices

A. Configuración del servidor

Para este punto voy a partir de una máquina con una instalación limpia de Ubuntu 14.04. El servidor sobre el que está corriendo la aplicación cuenta con las siguientes características hardware:

RAM 4GB

Disco duro 10GB

CPU Doble nucleo

A.1. Instalación de software

MySQL

```
sudo apt-get install mysql-server-5.6
```

Tomcat7

```
sudo apt-get install tomcat7
```

Instalación de Java

```
echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu precise main" |
tee -a /etc/apt/sources.list
echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu precise main" |
tee -a /etc/apt/sources.list
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EEA14886
sudo apt-get update
sudo apt-get install oracle-java7-installer
```

A.2. Configuración del software

MySQL - Acceso desde redes externas

1. Comentar la línea que comienza con 'bind-addressén el fichero /etc/mysql/my.cnf.
2. Reiniciar MySQL para que tome esta nueva configuración.
3. Por último permitimos la conexión desde cualquier IP:

```
GRANT ALL PRIVILEGES on *.* to root@'%' identified by 'root';
FLUSH PRIVILEGES;
```

Tomcat7

Modificación de la memoria RAM asignada a Tomcat 7 Para esta aplicación se va a aumentar la memoria RAM disponible. Con Tomcat7 se hace creando un fichero llamado 'setenv.sh'localizándolo en $CATALINA_HOME/bin. El contenido de este fichero son los parámetros con los que se quiera ejecutar Tomcat, en mi caso basta con añadir:

```
JAVA_OPTS="-Xms256m -Xmx1536m"
```

Permitir el acceso a tomcat-manager Para facilitar el despligue de aplicaciones web Tomcat dispone de un gestor web para manejar todas las aplicaciones que corran sobre nuestro servidor.

1. Por defecto el paquete que permite esta tarea no viene en la instalación de Tomcat 7 por lo que hay que instalarlo:

   ```
   sudo apt-get install tomcat7-admin
   ```

2. Para entrar en esta nueva pantalla se entra mediante un navegador:

   ```
   http://URL:8080/manager/html
   ```

3. Por defecto no hay configurado ningún rol y no se puede utilizar, la configuración de este servicio se encuentra en '/etc/tomcat7/tomcat-users.xml'y hay que incluir la siguiente linea para crear un usuario llamado ádmin'con contraseña 'password'para que acceda a esta funcionalidad:

   ```
   <user username="admin" password="password" roles="manager-gui,admin-gui"/>
   ```

A.3. Configuración del sistema operativo

Variables de entorno

```
JAVA_HOME=/usr/lib/jvm/java-7-oracle/jre
CATALINA_HOME=/usr/share/tomcat7
TFG_LOG=/var/log/TFG
TFG_PROGRAMAS=/opt/tfg
TFG_BACKUP=/opt/tfg/backup
```

Creación de directorios Hay que crear los siguientes diretorios:

```
/var/log/TFG
/opt/tfg/jar
/opt/tfg/scripts
/opt/tfg/backup
```

Tarea cron

```
0 2 * * * sh $TFG_PROGRAMAS/scripts/refresco.sh
```

B. Documentación entregada

El soporte físico entregado se compone de los siguientes archivos/directorios:

- memoria.pdf: Este archivo es la memoria del trabajo de final de grado.
- código fuente: Bajo este directorio se encuentran las versiones de código fuente de la aplicación, contiene los siguientes subdirectorios:
  - proyecto: En este subdirectorio se encuentra el código fuente de la última versión de la aplicación.
  - shellscript: Bajo este subdirectorio están los programas shellscript que se utilizan para las copias de seguridad o para el refresco de productos.
- ejecutables: Bajo esta carpeta se encuentra la página web, encapsulada en un archivo WAR (TFG.war) y el programa encargado de refrescar los datos periódicamente (TFG.jar).
- documentación adicional: Aunque a la hora de crear la memoria se ha intentado aumentar la facilidad de lectura, hay diagramas que son muy grandes y son difíciles de leer. Por este motivo se ha optado por adjuntar el archivo de la herramienta case que he utilizado (astah). Ese archivo está incluído en este directorio.

C. Configuración del log

Como framework para el log de la aplicación he decidido utilizar log4j en su segunda versión [39]. Decidí utilizar esta versión en vez de la primera porque me permite utilizar variables de entorno dentro del fichero de configuración del log. Gracias a esto no hay ninguna variación del fichero de configuración en el entorno de desarrollo y en el entorno de producción debido a los cambios en las rutas. En esta versión del framework no existe la posibilidad de filtrar los mensajes en cada appender por lo que he tenido que crear una clase que haga esta tarea e insertarla en log4j. La jerarquía del log es la siguiente:

- error.log: En este fichero van a estar disponibles todos los logs con nivel error de la aplicación, independientemente de la capa.
- info.log: En este fichero van a estar disponibles todos los logs con nivel info de la aplicación, independientemente de la capa.
- controlador: En esta carpeta van a estar disponibles los ficheros del log de la capa de controlador.
    - error.log: Fichero que contiene los mensajes del nivel de error de esta capa.
    - info.log: Fichero que contiene los mensajes del nivel de info de esta capa.
    - debug.log: Fichero que contiene los mensajes del nivel de debug de esta capa.
- modelo: En esta carpeta van a estar disponibles los ficheros del log de la capa de modelo.
    - error.log: Fichero que contiene los mensajes del nivel de error de esta capa.
    - info.log: Fichero que contiene los mensajes del nivel de info de esta capa.
    - debug.log: Fichero que contiene los mensajes del nivel de debug de esta capa.
- persistencia: En esta carpeta van a estar disponibles los ficheros del log de la capa de persistencia.
    - error.log: Fichero que contiene los mensajes del nivel de error de esta capa.
    - info.log: Fichero que contiene los mensajes del nivel de info de esta capa.
    - debug.log: Fichero que contiene los mensajes del nivel de debug de esta capa.