WebMovies

Integración de Sistemas
Aplicación Web en .NET

[FACULTADE DE INFORMÁTICA DA CORUÑA]

María Alicia de Andrés Herrero (maria.deandres)
Yago Méndez Vidal (yago.mendez.vidal)

[11/09/2012]

Contenido

Ilustraciones

1. Arquitectura global

WebMovies es una aplicación web diseñada e implementada con la plataforma de desarrollo y ejecución .NET, que no sólo nos brinda todas las herramientas y servicios que se necesitan para desarrollar modernas aplicaciones empresariales y de misión crítica, sino que también nos provee de mecanismos robustos, seguros y eficientes para asegurar que la ejecución de las mismas sea óptima.

Como entorno de desarrollo y ejecución de la aplicación usamos íntegramente Visual Studio 2010 Ultimate, creado para .NET que contiene todas las herramientas y módulos necesarios para llevarlo a cabo.

La versión del Microsoft .NET Framework usada es la 4.0, con la necesidad de tener instalada la 3.5 SP1 para retrocompatibilidad de una librería. El proveedor de acceso a la base de datos utilizado para la capa modelo que sirve como mapeador objeto-relacional, es el ADO.NET Entity Framework incluido en esta versión. Como gestor de base de datos empleamos SQL Server 2008 Express Edition, integrada en Visual Studio 2010 Ultimate. Para el desarrollo de la parte Web, tenemos ASP.NET y las librerías de la Microsoft Enterprise Library 4.1.

El servidor web, donde se hospeda la aplicación es el integrado en Visual Studio, ASP.NET Development Server, un servidor específico para albergar aplicaciones web ASP.NET, sin necesidad de desplegar en un Internet Information Server (IIS). El entorno de ejecución de la aplicación y de las pruebas se realiza sobre este servidor en local.

Usamos C#, desarrollado para .NET, como lenguaje para implementar los casos de uso del modelo y el código asociado a las páginas del proyecto Web. Todas las herramientas necesarias para un desarrollo cómodo y completo se incluyen en Visual Studio.

La herramienta de modelado UML empleada para los diagramas es la que se incluye en el Visual Studio 2010 Ultimate y como cliente para el repositorio de control de versiones durante la etapa de desarrollo se ha empleado AnkhSVN 2.X.

Los sistemas operativos sobre los que se han desarrollado y ejecutado la aplicación son Windows 7 para ambos componentes del grupo.

WebMovies es una solución que consta de cuatro proyectos que se muestran en la figura siguiente (Ilustración 1.1): Model, de tipo Class Library; Test, de tipo Test Project; Web, de tipo ASP.NET Web Application; y Doc, de tipo Modeling Project. Todos ellos están englobandos en el mismo espacio de nombres `Es.Udc.DotNet.WebMovies`.

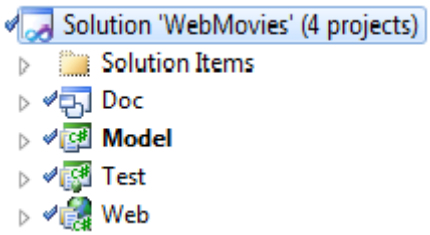

Ilustración 1.1. Estructura de la solución

1.1. Model

El primer proyecto de la solución, Model (Ilustración 1.2), corresponde con la capa modelo. Esta capa se encarga de implementar el acceso a los datos y la lógica de negocio. Los casos de uso que define se presentan en los 8 servicios en los que dividimos la lógica de la aplicación, lo cual conforma una API que proporciona toda la funcionalidad necesaria, completa e independiente para las capas superiores.

Para su desarrollo seguimos los mismos pasos que en el desarrollo de la capa modelo para toda aplicación empresarial: primero modelaremos las entidades (capa de persistencia), creamos una funcionalidad que permita gestionar la persistencia (DAOs), definimos los casos de uso (servicios) y diseñamos las pruebas de unidad.

En la carpeta `Sql`, tenemos `SqlServerInitializeDataBase.sql`, un script que contiene los comandos para la inicialización de la base de datos, la creación de las tablas y la inserción de los datos básicos en inmutables para un correcto funcionamiento. En un entorno local, con la instancia de SQL Server 2008 Express corriendo, se crea la base de datos y el log configurados en este fichero, al igual que el usuario requerido en la connection string[1].

[1] Para más información sobre configuración de la base de datos consultar documentación del fichero SQL en los archivos de la aplicación.

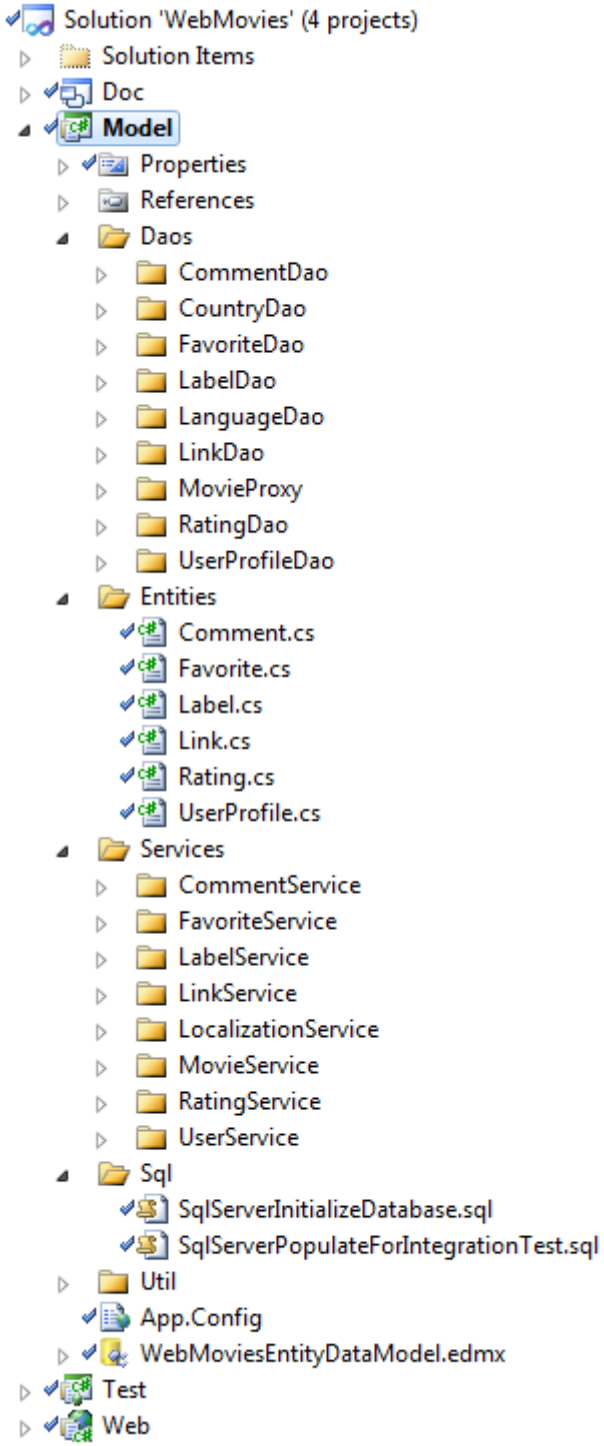

Ilustración 1.2. Estructura del proyecto Model

Adicionalmente, en la carpeta `Sql` se incluye un segundo fichero llamado `SqlServerPopulateForIntegrationTest.sql`, que proporciona unos datos elementales para una prueba de integración y evaluación de la aplicación web sin tener que crearlos cada vez que se genere, muy útil durante la etapa de desarrollo.

Una característica del Entity Framework es la posibilidad de generar un el modelo en el lenguaje de programación a partir de las tablas empleando un enfoque DataBase First, dando la posibilidad de la generación automática de las clases persistentes con el Entity Data Model Wizard, un asistente que utiliza la base de datos creada para modelar dichas clases, obteniendo así el fichero que se ve en la Ilustración 2.1, llamado `WebMoviesEntityDataModel.edmx`. Se trata de un fichero XML que contiene las definiciones de los esquemas: conceptual (CSDL), de almacenamiento (SSDL) y las relaciones entre ambos (C-S). Además en su misma carpeta, encontramos un fichero generado también automáticamente, que es una clase parcial asociada al `.edmx`, con la extensión `.edmx.Designer.cs`.

Se puede ver también en la figura, una carpeta llamada `Entities`, que contiene las clases parciales asociadas a las clases de entidades persistentes, que fueron creados con el fin de redefinir los métodos `Equals()`, `GetHashCode()` y `ToString()`, de especial utilidad para los tests.

La carpeta `Dao` contiene la interfaz e implementación de los objetos de acceso a datos para cada una de las entidades correspondientes. Los DAOs se pueden implementar de múltiples maneras, ya sea a través de consultas SQL o empleando las características del Entity Framework de .NET, aproximación que será la empleada en esta práctica. Se explicará el funcionamiento de los DAOs en un apartado posterior. Cabe destacar que adicionalmente se incluye aquí el proxy de acceso al servicio web externo ***WebShop*** que, aunque no en el sentido clásico de un DAO, sí que es el objeto que ofrece el acceso a los datos.

En cuanto al directorio `Services`, alberga las interfaces e implementación de cada uno de los grupos de casos de uso identificados, relacionados entre sí y conformando cada uno de los servicios, que constituyen en su conjunto la API para la lógica de negocio de nuestra aplicación. También se comentarán éstos en secciones posteriores.

Nos queda mencionar la carpeta `Util`, que contiene diferentes utilidades para la implementación de la capa modelo:

- `PasswordEncrypter`: Clase que contiene un algoritmo de encriptación empleado para almacenar y validar las contraseñas de usuario.
- `Locale`: se trata de un custom value object para representar los pares `Language-Country`. Se incluye aquí y no como un posible `LocaleDetails` del servicio de localización porque no es un elemento que se emplee en ese servicio, sino cómo una utilidad gestionar los datos obtenidos de éste.
- `Exceptions` y `Property`: Detectamos dos posibles mejoras en la gestión de excepciones de `ModelUtil`: una era poder identificar qué entidad es la afectada en caso de fallo y la otra era poder identificar qué propiedades se intentaban recuperar en caso de que no fuese un identificador concreto. Por ello hemos extendido las excepciones provistas en `ModelUtil` haciéndolas genéricas y proveyéndolas de un mecanismo de almacenamiento de propiedades que se intentaban cumplir. Así si intentásemos acceder a los comentarios de un usuario (1) para un enlace (2), podríamos obtener:
  - `InstanceNotFoundException<Comment>` userId=1, linkId=2;
  - `InstanceNotFoundException<UserProfile>` userId=1; o
  - `InstanceNotFoundException<Link>` linkId=2

  pudiendo ser capturadas cada una de forma. Cabe destacar la gran utilidad a la hora de hacer las pruebas unitarias, ya que, por ejemplo, un podría esperarse `IntanceNotFoundException`, pero realmente en vez de afectar a un `Link` estuviese afectando a un `UserProfile`.
- `Dao`: Una extensión del `GenericDao` de `ModelUtil` para proveer de forma directa las nuevas excepciones y además lanzar la `InstanceNotFoundException<E>` en el método `Update()`, cosa que no se contempla en el original.
- `Collections`: En esta carpeta se presentan tres clases con bloques de diferentes colecciones usadas en la aplicación, concretamente `DictionaryBlock`, `ListBlock` y `SortedListBlock`. Estas clases son extensiones de las clases genéricas de la API de .NET para añadirles las propiedades `Index` y `HasMore` y así proveer uan estructura de datos cómoda para los listados por bloques y que de forma transparente sigan siendo objetos de la API.

Por último, el fichero `App.Config` guarda la configuración de este primer proyecto, donde se configura el logginApplicationBlock, el contenedor Unity, las cadenas de conexión a las bases de datos y algunas propiedades generales de la aplicación.

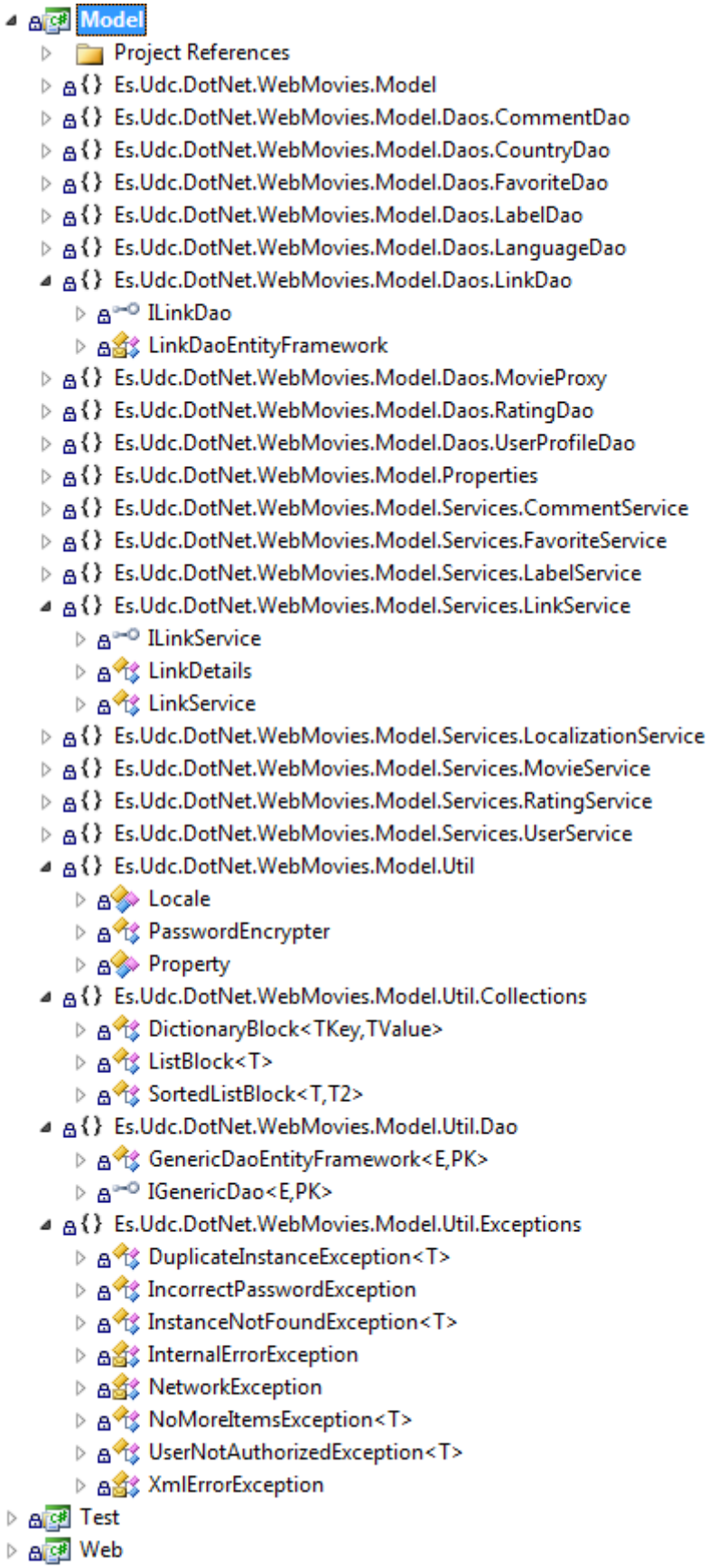

Ilustración 1.3. Clases del proyecto Model

1.2. Test

En el proyecto Test (Ilustración 1.4), se accederá al proyecto Model para poder realizar las pruebas para cada caso de uso de los servicios. Para ello, se crea una clase para cada servicio de la capa modelo (carpeta Test), en el cual se probarán los casos de uso de manera individual e independiente al resto. Por cada caso de uso se contempla un método para el funcionamiento normal, fraccionado en etapas en caso de ser un listado para comprobar la navegación por bloques así como el orden; y un método por cada uno de los casos de error.

Antes y después de ejecutarse la clase que contiene los test de cada servicio, se realizan una serie de operaciones de inicialización y finalización de clase en donde se instancia el contenedor de Unity y le hace dispose(), respectivamente. Asimismo, se definen los métodos de inicialización y finalización de prueba para la gestión de cada transacción, a la cual se hará rollback para evitar que los datos queden permanentemente registrados. Estos métodos se pueden ver al principio de cada una de las clases de test, y están documentados. Cabe hacerse notar que estos métodos han sido modificados para soportar la característica que se comenta a continuación.

En la carpeta `Util` se recoge el `TestManager` provisto para la elaboración de la práctica así como una nueva clase `TestUtil`. Los DAOs han sido retirados de los tests e introducidos en esta nueva clase (solamente se instancian la primera vez que son necesarios) que provee métodos de creación de entidades de prueba predefinidas sin tener que ser instanciadas de forma individual en cada método, añadiendo además un rodeo para evitar problemas con las fechas de forma transparente. Los datos de prueba se almacenan como propiedades en las `Settings` del proyecto y son cargadas en esta clase como una propiedad llamada `TestData`. Además se proveen unos nuevos métodos de aserción de equivalencia para comparar entidades con sus detalles de forma cómoda y sencilla. Cabe mencionar que en el desarrollo de esta clase de utilidades primero se optó por una aproximación de métodos estáticos que falla, ya que cada prueba debe ser ejecutada con un único contenedor de Unity con el cual instanciar los DAOs.

Finalmente en `XmlDocuments` se recogen los ficheros que ha de devolver el servicio de películas desde el servicio web externo para los métodos que no parsean las respuestas.

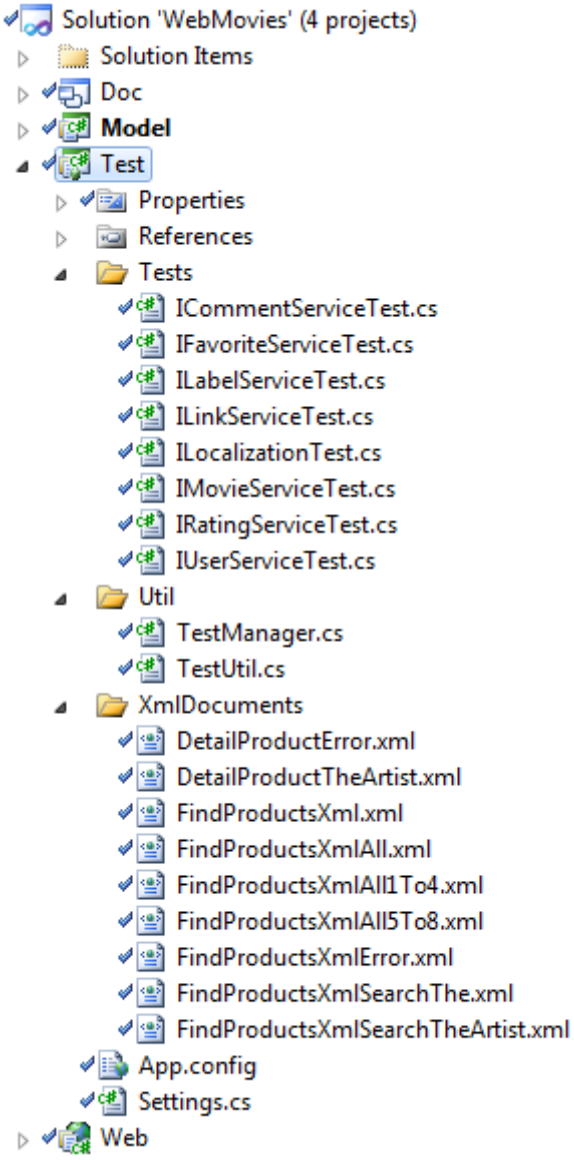

Ilustración 1.4. Proyecto de test

1.3. Web

El último proyecto de la solución es el Web. Este proyecto será el principal de la solución y ofrecerá una interfaz gráfica para poder hacer uso del modelo mientras interactúa con nuestro primer proyecto Model. Recordemos que nuestra aplicación web se ejecuta y carga dinámicamente en el servidor SQL Server de nuestra máquina.

El directorio `Http` contiene cuatro directorios:

1. `Session`:
    a. `SessionManager`, que establece propiedades de cada usuario en la sesión y sus valores se mantienen durante la misma. Es una fachada que maneja los session objects y las cookies. Contiene métodos invocables únicamente con fines útiles para la parte vista tales como operaciones específicas de la parte vista para registro de usuarios, etc.
    b. `SpecificCulturePage`, extensión de `System.Web.UI.Page` que contiene los métodos necesarios para establecer la localización para la página web. Si el usuario se encuentra autenticado se empelará la combinación del lenguaje y país seleccionados en la página de perfil de usuario; o si no, la obtenida por la configuración del navegador y el sistema operativo. Hace uso las clases Countries y Languages (explicados posteriormente) empleados para mostrar la colección de países y lenguajes, respectivamente.
    c. `UserSession`, representa las propiedades necesarias para la navegación autenticado que han de estar presentes en el servidor. Están gestionadas por el `SessionManager`.
2. `Util`: contiene la clase `CookiesManager` que, como su nombre indica, gestiona aquellas propiedades que han de ser almacenadas en la parte del cliente en forma de cookie del navegador, incluyendo el nombre de usuario, la clave encriptada y el tipo de búsqueda empleada: normal o avanzada.

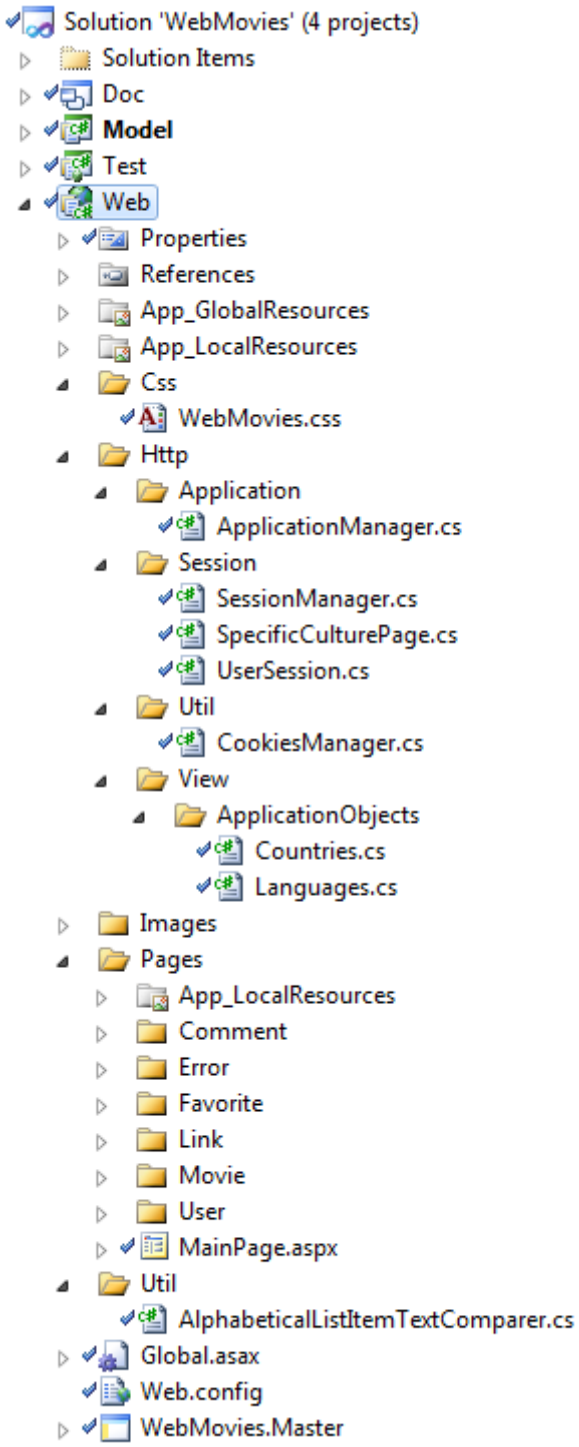

Ilustración 1.5. Vista del proyecto Web

3. `Application`: contiene la clase `ApplicationManager` que gestiona aquellos datos que han de ser manejados de forma global para toda la aplicación, obviamente a nivel de servidor. Esta clase ha recibido una gran cantidad de trabajo para permitir que implemente la resolución de títulos de películas para todas las páginas de la aplicación haciendo caché de todos aquellos que han sido recuperados y así evitar acceder muy repetidamente al servicio web externo.
4. `View/ApplicationObjects`: con las clases `Country` y `Language` explicadas anteriormente recuperan de la base de datos aquellos países, lenguajes y combinación de ellos que soporta la aplicación. Cabe destacar que estas clases han recibido una gran revisión permitiendo recuperar la información de la base de datos, localizar las cadenas y ordenarlas alfabéticamente, empleando la clase de utilidad `AlphabeticalListtemTextComparer` y un `SortedSet`.

Las carpetas `Css` e `Images` contienen la hoja de estilo y las imágenes necesarias para la renderización de las páginas web. Cabe destacar que las imágenes siempre se accederán en forma de recursos y que en la hoja de estilo se almacena información relevante, no simplemente estilo, ya que por ejemplo el realce o degradado de los enlaces promovidos o denunciados, respectivamente, se hace empelando estas propiedades.

La carpeta `Pages` contiene las páginas ASP por las que se navegará. Cada una de las páginas está compuesta por un trio de ficheros: el `.aspx`, un markup con el código necesario para generar el XHTML; el `.aspx.cs`, con el code behind en C#, que implementa todo aquel código necesario de acceso a los servicios; y un `.aspx.designer.cs`, con las propiedades que representan cada uno de los controles de ASP en la página. Adicionalmente existen uno o varios ficheros de recursos en el `App_LocalResources` de cada directorio con ficheros `.resx` para cada uno de los cultures soportados para esa página.

La nomenclatura de cada una de las páginas hace referencia a la entidad que presentará siendo éstas:

- Entity.aspx: muestra esa entidad
- AddEntity.aspx: añade una nueva instancia de esa entidad al sistema

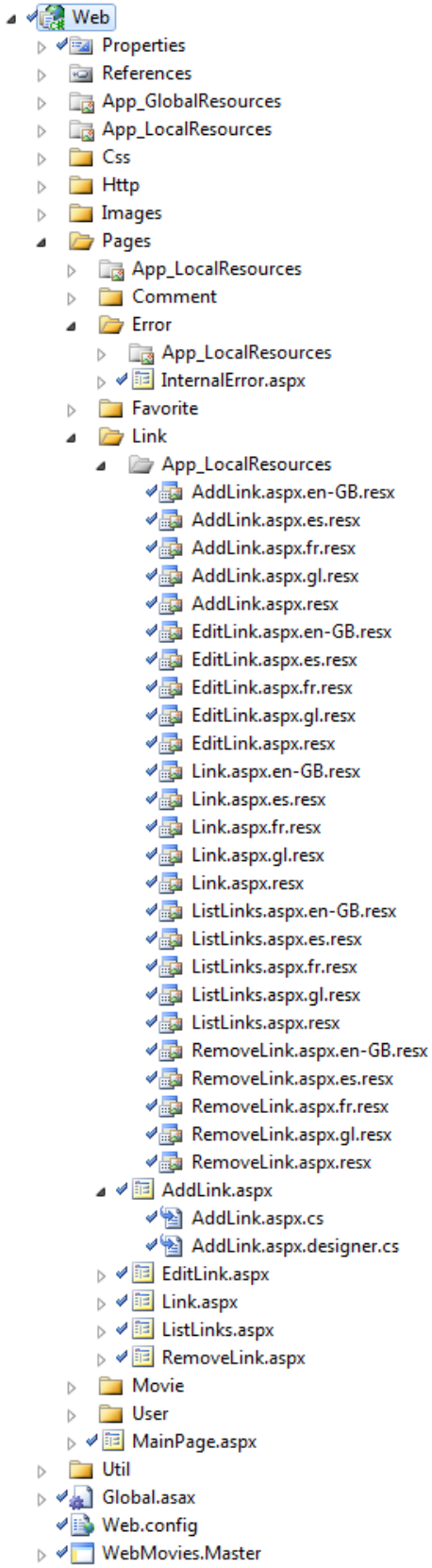

Ilustración 1.6. Vista del proyecto Web por clases

- EditEntity: edita una instancia existente de dicha entidad
- RemoveEntity: meustra un diálogo de confirmación para eliminar una instancia de la entidad
- ListEntities: empleando un ListView se presenta una vista simplificada de las entidades ordenadas en forma de lista paginada.

Los parámetros necesarios para cada entidad se pasarán como parámetros de la request HTTP y variarán según las necesidades.

En el `AppGlobalResources` se recogen 3 grupos de resources que estarán accesibles para toda la aplicación: `Common`, con los textos generales de toda la aplicación, incluyendo el título y el logo; `Countries` y `Languages`, que recogen los nombres de estos en cada uno de los locales soportados.

En el raíz del proyecto se encuentran el `Global.asax`, conde se instancia el contenedor de Unity y se ejecutan los métodos ahí definidos en respuesta a eventos de toda la aplicación; la WebMovies.master, nuna plantilla en la que se apoyan todas las páginas de la aplicación; y el fichero `Web.config`, con definiciones similares a las del App.config del proyecto Model en cuanto a cadenas de conexión, log, y Unity, así como añadir propiedades específicas para controlar el acceso a los diferentes recursos para usuarios anónimos y autenticados, sistema de autenticación… y demás configuraciones.

1.4. Doc

Aunque este proyecto no forma parte de lo necesario para la solución se ha decidido crear este documento para desarrollar la documentación asociada al trabajo, albergar notas para el desarrollo, realizar la memoria, almacenar las imágenes generadas… y tener un control de versión sobre el trabajo realizado.

Los diagramas han sido realizados empleando las herramientas que proporciona Visual Studio. Se tienen los diagramas de los diferentes servicios, así como dos diagramas de secuencia, uno con un caso de uso del modelo y otro donde se refleja la interacción en la interfaz Web, y algún diagrama de clases más que se hicieron durante el desarrollo de la práctica. Además hay otras carpetas donde se encuentran imágenes, notas y el documento de la memoria.

Los diagramas de clase se encuentran en el proyecto Model por restricciones de VisualStudio.

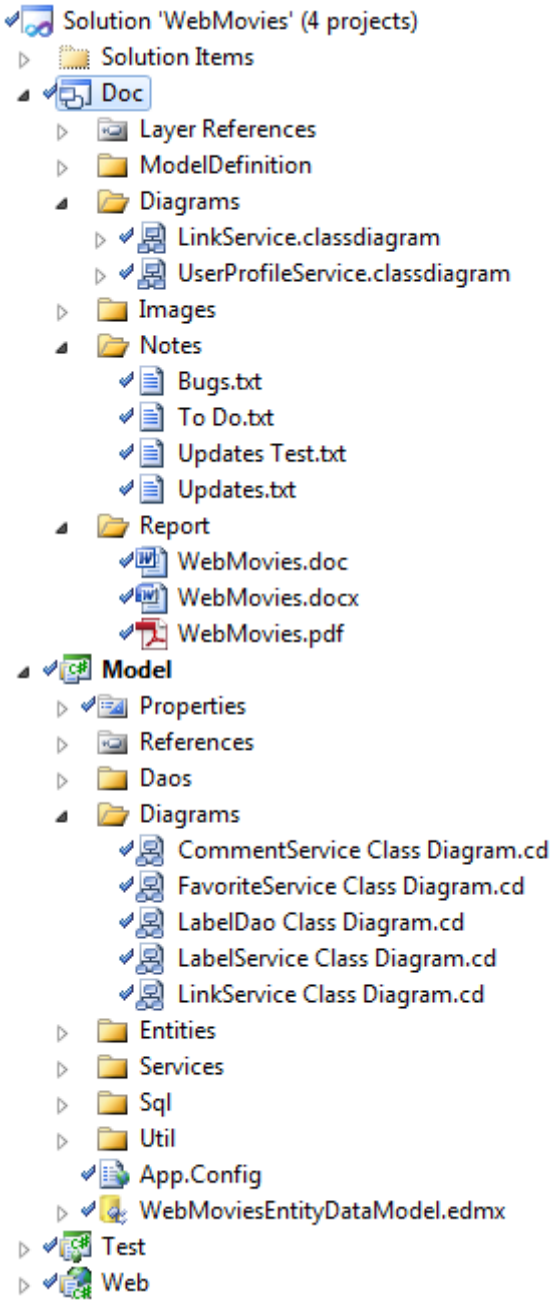

Ilustración 1.7. Doc, proyecto de modelado y documentación

2. Modelo

2.1. Clases persistentes

En la siguiente, figura 2.1 se muestra el diagrama de las clases persistentes, creado a partir de la herramienta Entity Data Model Wizard en un archivo con extensión .emdx. Como se ha explicado anteriormente, hemos generado automáticamente las clases a partir del script SQL con las entidades del modelo relacional, es el Entity Framework quien hace este mapeo (ORM).

Además de generarse este diagrama con extensión .edmx, se crea una clase parcial asociada al mismo. Se trata de un fichero xml con extensión .edmx.Desginer.cs que contiene las entidades del modelo conceptual (CSDL), el (SSDL) o esquema de la estructura física de la BD y por último el (MSL) que contiene las relaciones entre las distintas clases del modelo conceptual con las tablas del físico.

Con Entity Client podremos trabajar con estos modelos (generados con el Entity Wizard) en vez de hacerlo directamente con la base de datos, consiguiendo así una mayor independencia.

Entre las relaciones relevantes del modelo, cabe destacar que hemos reflejado en la base de datos los lenguajes y países que se configurarán para cada sesión, manteniendo el valor de la variable asociada a la cultura del navegador, según lo seleccionado por el usuario o bien por defecto. La relación es muchos a muchos, un país puede hablar varios lenguajes, un lenguaje se puede hablar en varios países.

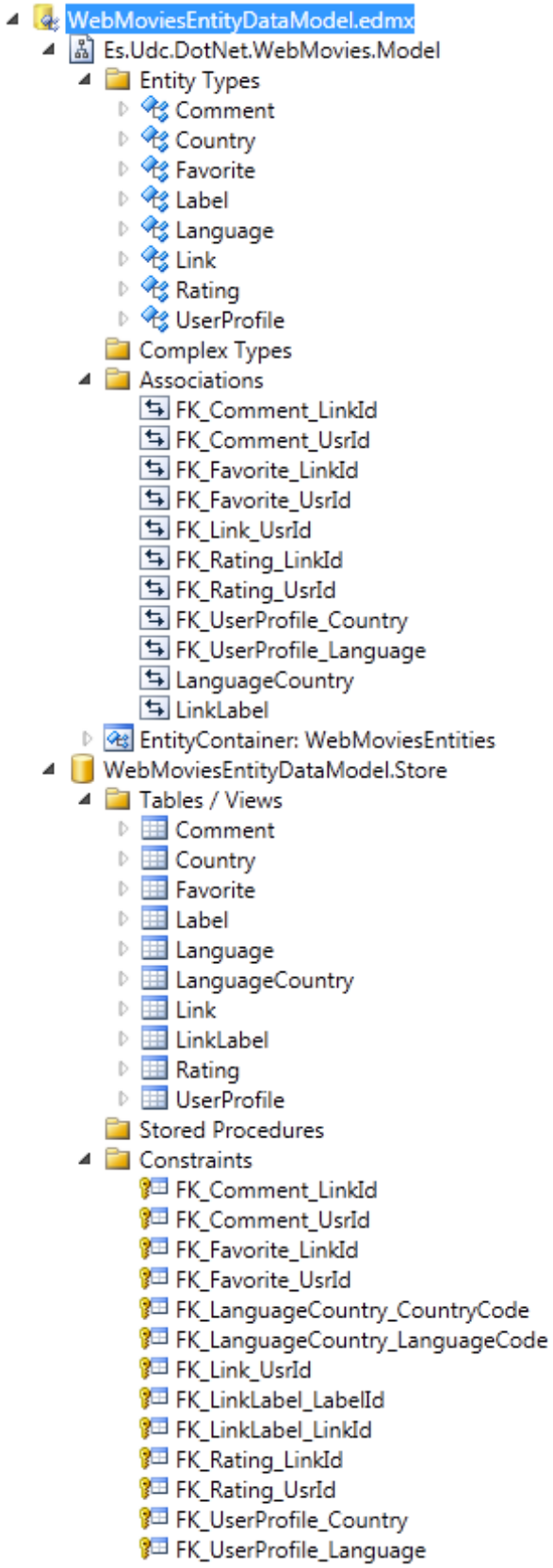

Ilustración 2.1. Contenido del .edmx

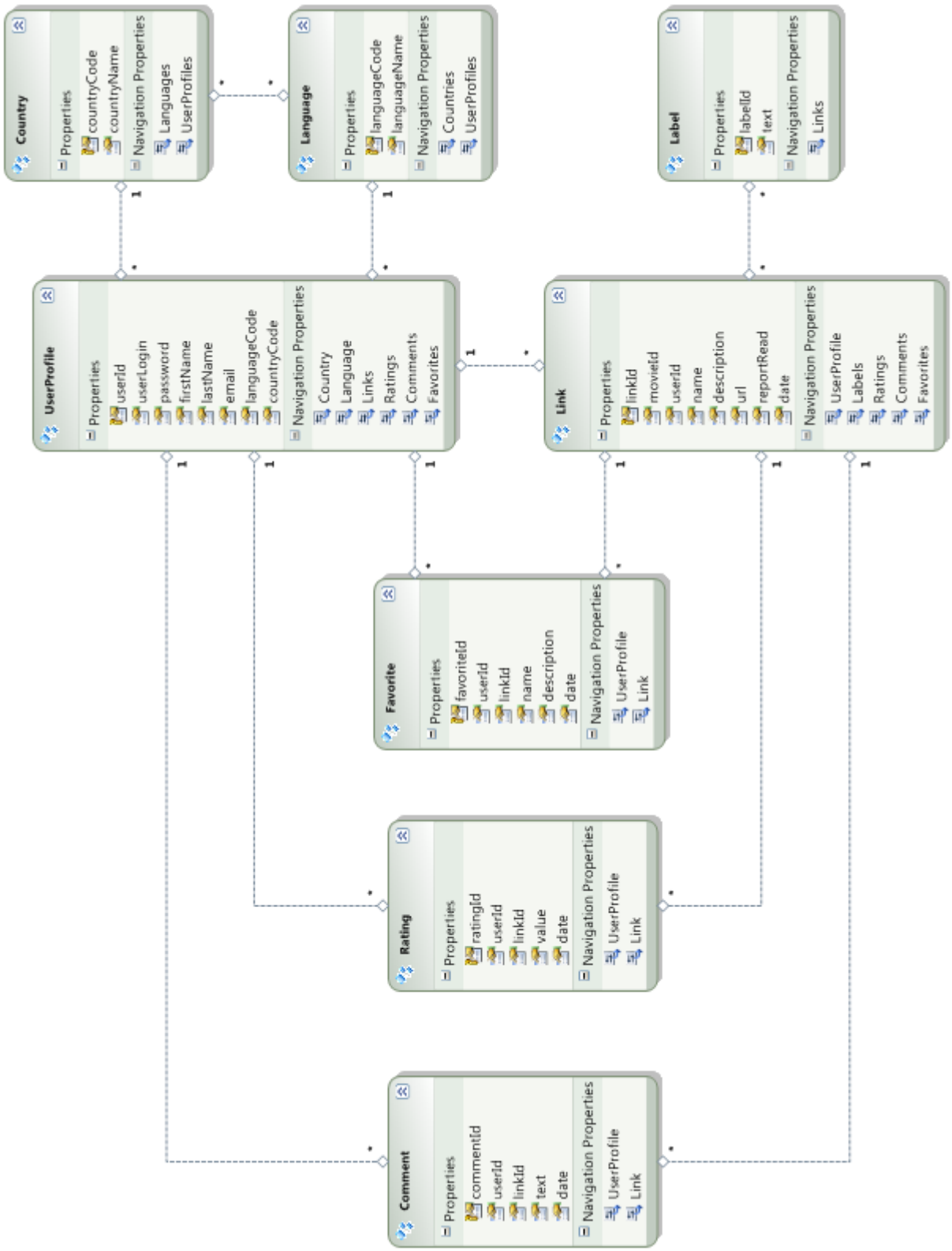

Ilustración 2.2. .edmx

2.2. Interfaces de los servicios ofrecidos por el modelo

Finalmente, se ha decidido tener 8 servicios, en el proyecto, asegurando un bajo acoplamiento.

Los servicios de comentarios, votos, favoritos y etiquetas se han desarrollado como servicios satélite conectados a los servicios de usuario y enlaces. De este modo los cambios para añadir una nueva entidad, por ejemplo imágenes, a la que poder aplicar comentarios, votos, favoritos y etiquetas requerirían simplemente ampliar los servicios disponibles sin tener que alterar en nada el funcionamiento de los demás. Con este planteamiento conseguimos un bajo acoplamiento, independencia entre los sistemas y, por tanto, una gran potencia de ampliación.

A continuación se muestran los servicios, en este caso los nombramos apenas, ya que, están documentados en el código cada uno de los métodos que implementan los casos de uso, por lo que no vamos a volver a explicarlos aquí.

`UserService`: Este servicio agrupa los casos de uso relativos a la gestión de la información de los usuarios.

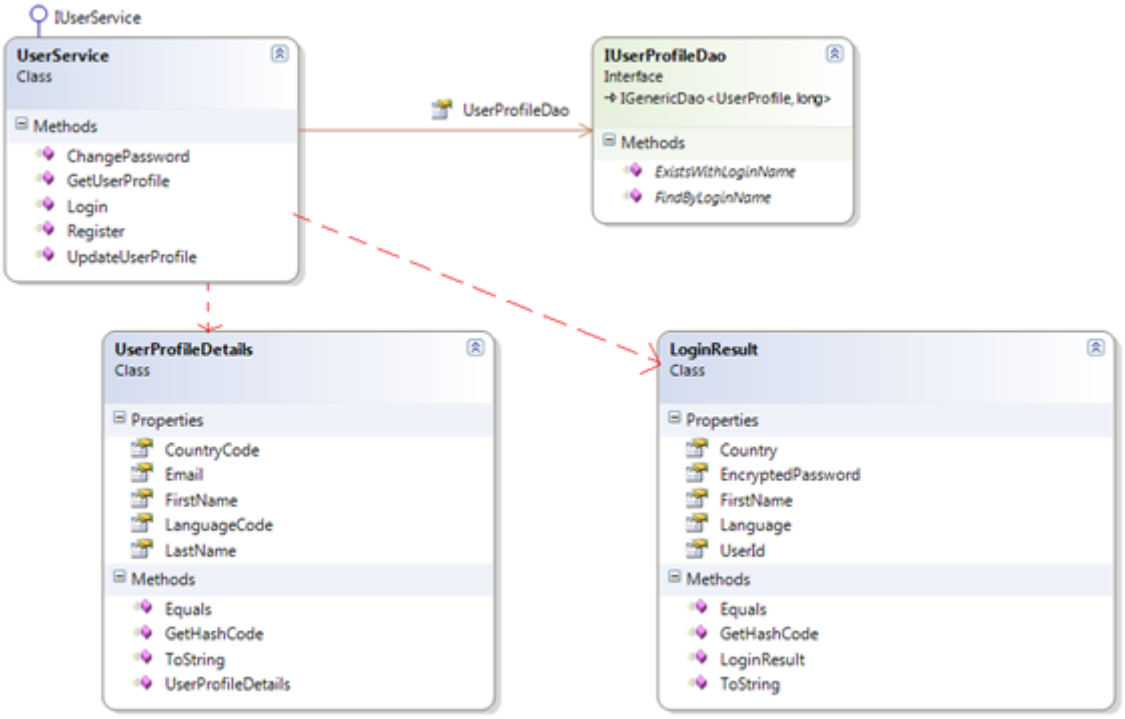

Ilustración 2.3. UserService

`FavoriteService`: Agrupa casos de uso relativos a la gestión de favoritos.

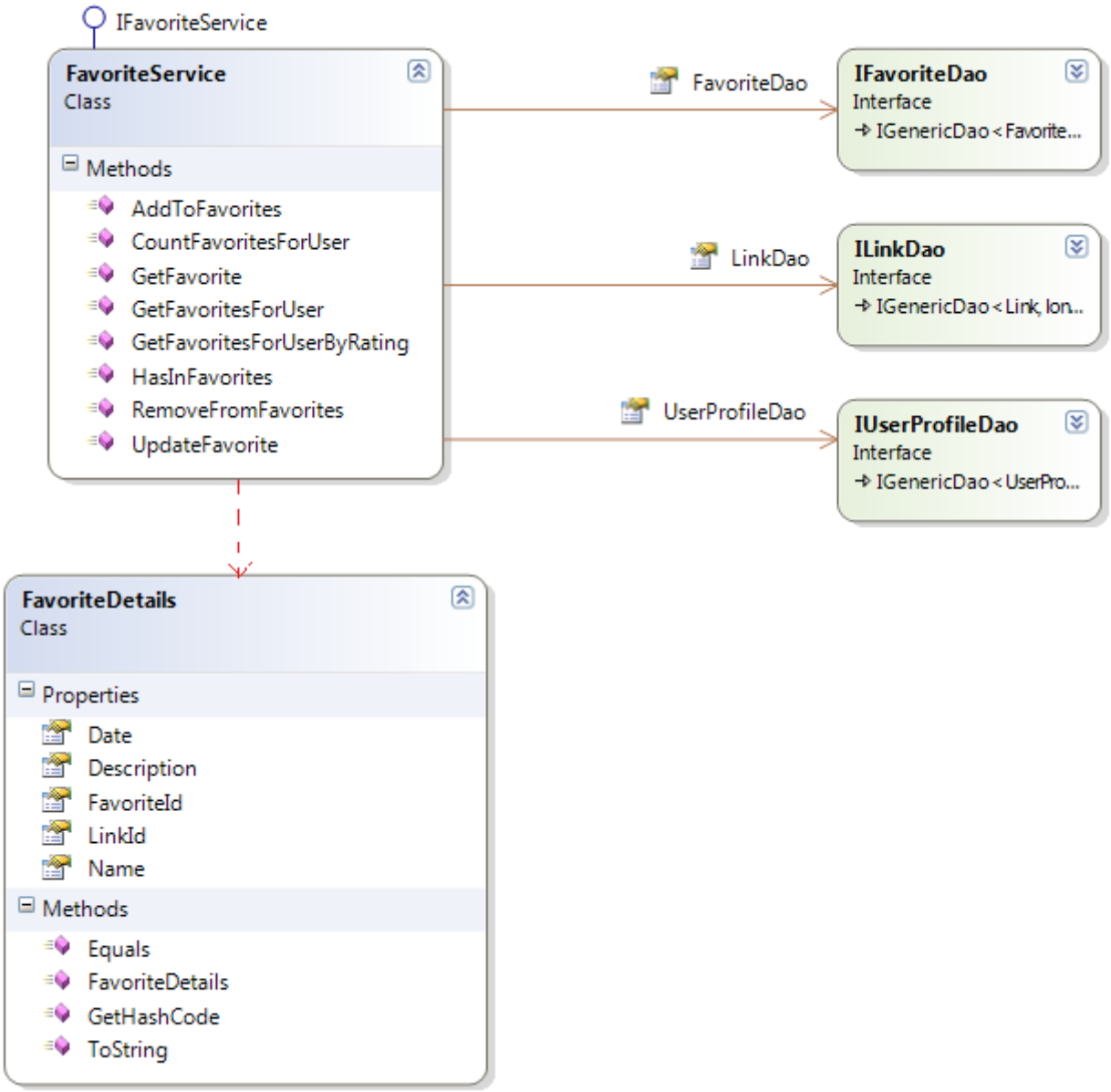

Ilustración 2.4. FavoriteService

`LabelService`: Agrupa casos de uso relativos a la gestión de labels.

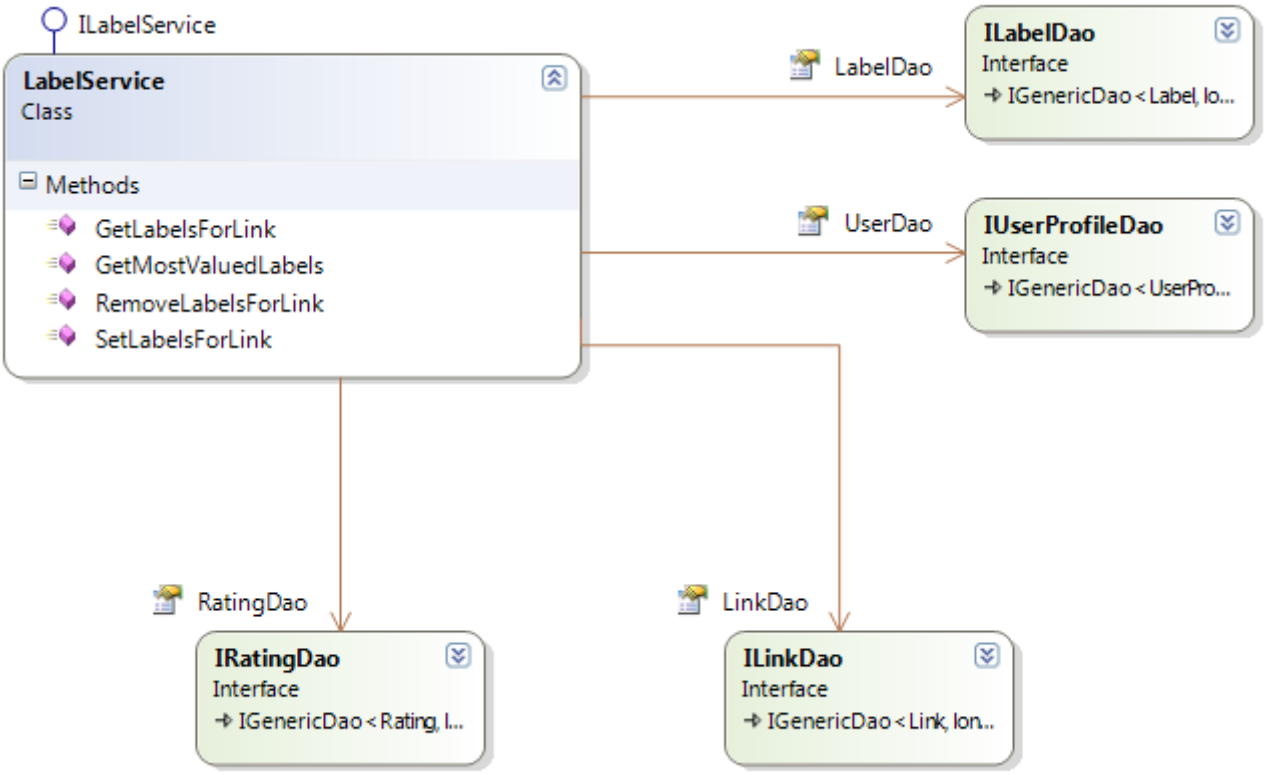

Ilustración 2.5. LabelService

`LocalizationService`: Agrupa los casos de usos que nos sirven para la gestión de la internalización deseada por el usuario para la navegación.

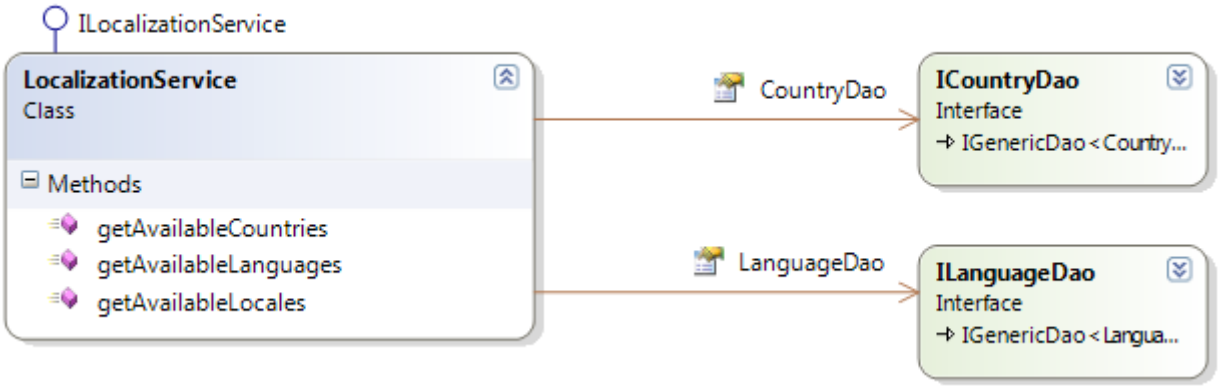

Ilustración 2.6. LocalizationService

Figure 2.6: Localization Service

`RatingService`: esta fachada contiene los casos de uso relativos a la gestión de las valoraciones sobre links por parte de los usuarios.

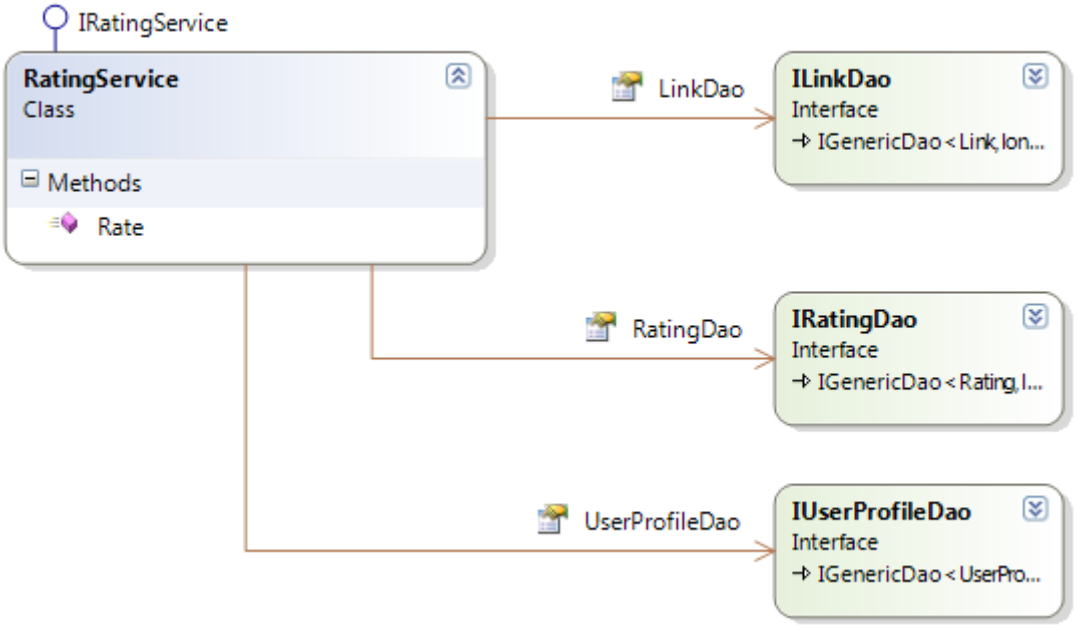

Ilustración 2.7. RatingService

`MovieService` : Esta fachada contiene los casos de uso relativos a la gestión de las películas, que será realizada por los usuarios.

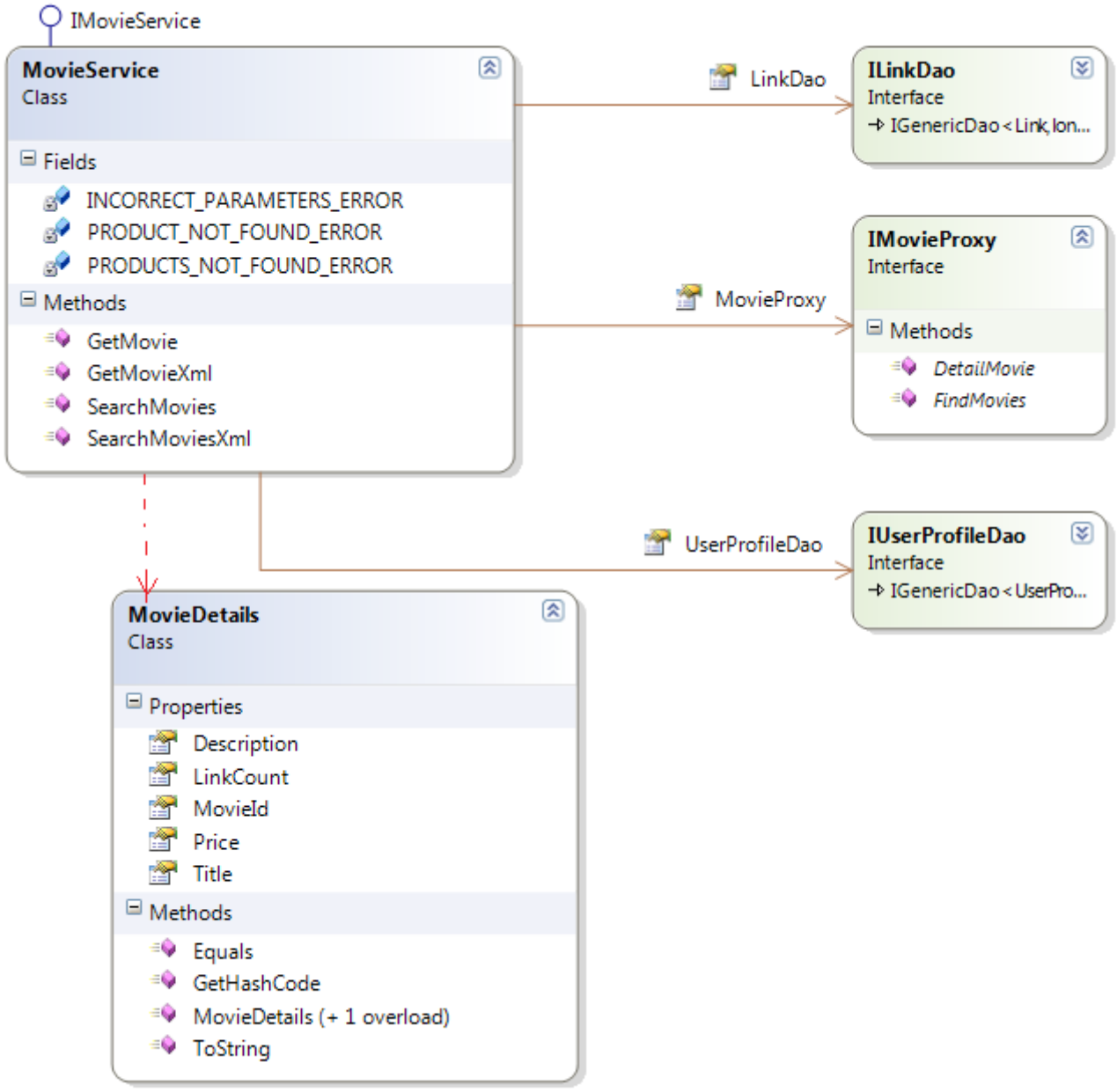

Ilustración 2.8. MovieService

Observar que dentro de este servicio, están contenidos los casos de uso que parsearán el XML para la integración con la aplicación desarrollada en la primera parte de la asignatura.

La justificación para crear este servicio en esta capa modelo, que tan solo lleva dos casos de uso asociados (cuatro incluyendo la parte optativa), es que se trata de una fachada que ofrece un servicio web y necesitamos tratar con los datos que obtenemos de este servicio en esta misma capa, pues el movieId se persiste en nuestro modelo. Se ha creado una clase movieProxy para acceder a ese servicio, ya que si no se ejecutase en local y tuviésemos la necesidad de salir a otro equipo o hacia internet, es importante tener lo más desacoplado posible todo aquello que dependa de una aplicación externa a la nuestra, ya que los cambios nos afectarán y puede ser muy costoso tratarlo.

En este caso, la implementación es muy sencilla con esta fachada obtenemos datos de la otra aplicación Web a partir de su URL y los devolvemos en forma de String.

CommentService: En él se implementan los casos de uso relacionados con los comentarios.

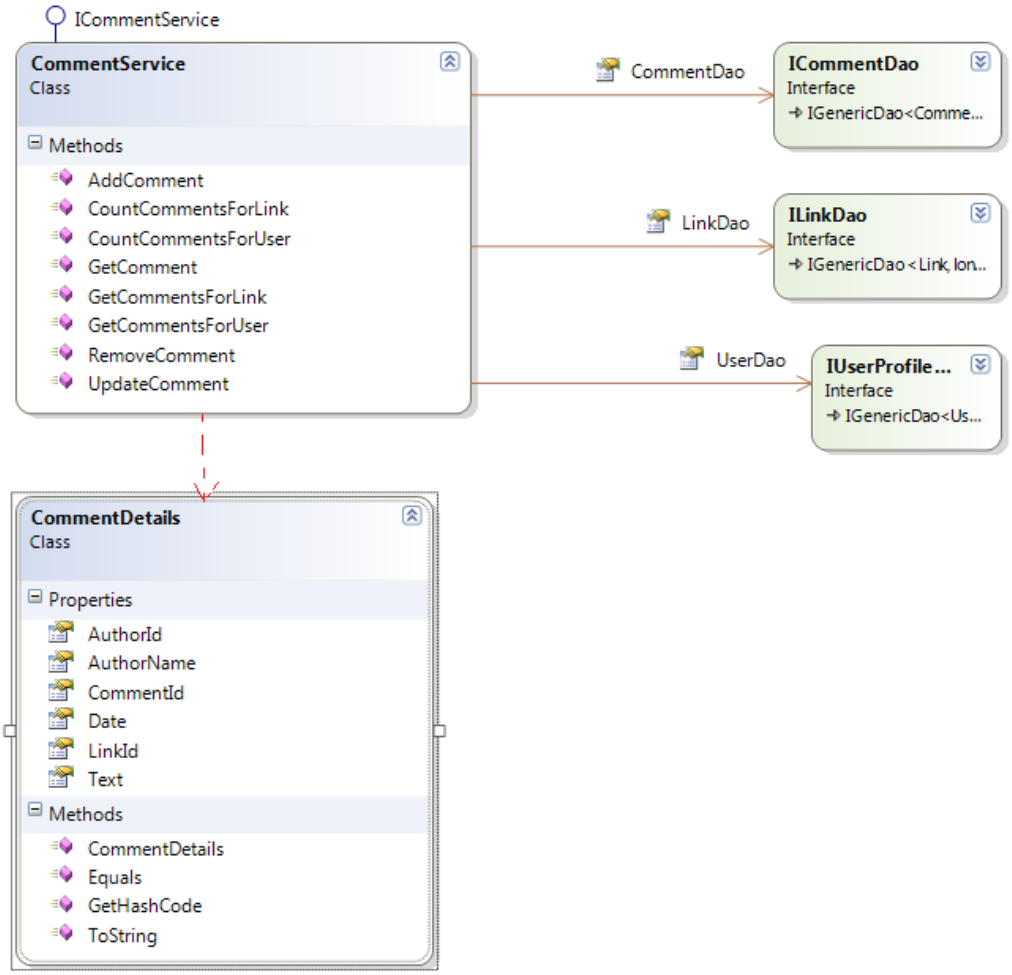

Ilustración 2.9. CommentService

`LinkService`: Agrupa casos de uso relativos a los links. Este servicio se explicará con mayor detenimiento en apartados posteriores.

2.3. Diseño de un DAO

En la siguiente figura, se puede ver un diagrama de clases que representa la implementación de un DAO. Como ejemplo usamos el LabelDaoEntityFramework, siguiendo el resto de los Daos una arquitectura similar. Para esta práctica y la implementación para EntityFramework se ha usado en todos los daos LINQ, debido a su simplicidad ya que nos ofrece corrección en tiempo de compilación.

Un Dao ofrece una interfaz de acceso a datos de la base de datos para que los

servicios sean totalmente independientes de las bases de datos y no necesiten conocerlas.

En los ficheros de configuración se indicarán que implementaciones concretas de los daos se mapearán a las interfaces. Como se puede ver en la imagen, existe un DAO genérico intermedio que extiende del IGenericDao implementado en el modelutil, nuestras interfaces para los DAOs heredan de ese DAO genérico, que es una interfaz que tiene las firmas de los métodos básicos de acceso a base de datos (CRUD) y cuya moneclatura se comenta al final de este apartado. El motivo por el que se creó este nuevo DAO del que extienden el resto de interfaces de los DAOs concretos para las entidades del modelo, es para generar excepciones parametrizadas y para mantener la coherencia lógica, aunque esto se explicará con detenimiento en el apartado de otros aspectos.

Finalmente de las interfaces de los DAO's (no importa en que nivel de la jerarquía) ,se extienden las clases concretas, XXXDaoEntityFramework. Estas clases son las que implementarán los métodos adecuados, en los que se encapsulan las consultas a la base de datos. Si nos referimos al último nivel de esta jerarquía, estos daos sirven como capa de acceso a la base de datos, desde cada una de las clases de nuestro modelo conceptual, en los otros casos implementan las operaciones tipicas de acceso a la base de datos que heredarán las clases en las que se realizan.

Con este diseño, si hay que portar la aplicación a otro sistema con una base de datos distinta, bastaría con realizar nuevas implementaciones para esa base de datos concreta y configurar el mapeo de las interfaces en los ficheros de configuración.

Finalmente en los DAO se siguen las siguientes normas de nombrado:

- Create: crea la entidad e inicializa la clave.
- Exists: devuelve un dato booleano indicando si existe entidad con el identificador o propiedad indicada.
- Count: indica el número de entidades que existen en el sistema
- Update: actualiza la entidad y lanza una excepción en caso de que no exista.
- Remove: elimina la entidad indicada y lanza una excepción si no existe.
- Find: recupera una entidad o una colección de ellas por la propiedad indicada, o su clave si no se especifica, y lanza una excepción de instanciación si no existe ninguna o si existe más de una.

- List: recupera un bloque de elementos de la entidad, que estará vacío si no existe ningún elemento sin lanzar ninguna excepción.
- ByProperties (WithEntities en el caso de Exists-): indica que la operación se realiza en aquellos elementos que respeten esos valores indicados para esa propiedad.
- ForEntities: indica que la operación se realiza limitandola sobre las claves foráneas de esas entidades referidas.
- All: indica que se aplica la operación a todas las entidades que maneja ese DAO.

2.4. Diseño de un servicio del modelo

Para este servicio y los demás, los métodos están nombrados con lenguaje natural de forma lo más sencilla y clara posible, pero se cumplen las dos siguientes reglas para la firma:

- Todos los parámetros son tipos primitivos o una colección de ellos.
- Si la respuesta no es un tipo primitivo, nunca se devuelve una entidad, sino que será un custom value object con los datos relevantes.

Además, todas las listas y bloques se asumen ordenados por fecha a no ser que el método lleve el sufijo -Rated, en cuyo caso la ordenación se hace por valoración.

Para devolver bloques de datos es adecuado además de la colección añadir información adicional como el índice en el cual empieza el bloque, si existen más elementos a continuación en la colección completa o el número total de elementos.

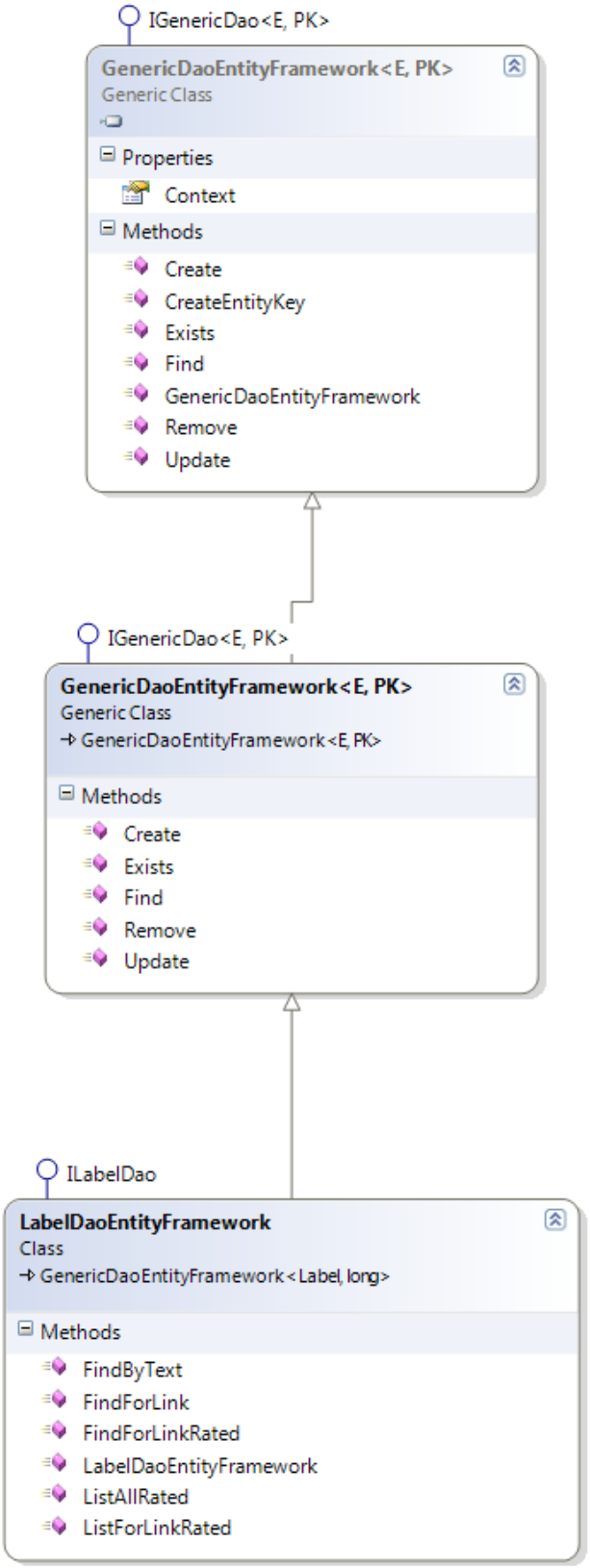

Ilustración 2.10. Diseño de un DAO

Para ello se han extendido las clases List y SortedList (pares clave-valor ordenados) de la API de Microsoft para añadirle esta información a través de propiedades, así como nuevos constructores para permitir añadir esta información de forma cómoda.

De este modo para una llamada a un método que espere una List será transparente, ya que el objeto devuelto es una List, como es el caso del GridView; pero para una llamada que espere un ListBlock, puede explotar sus características adicionales.

Las clases List y sortedList no aparecen en el diagrama de abajo pero simplemente, serían las clases importadas por ILinkService, para devolver la lista de links.

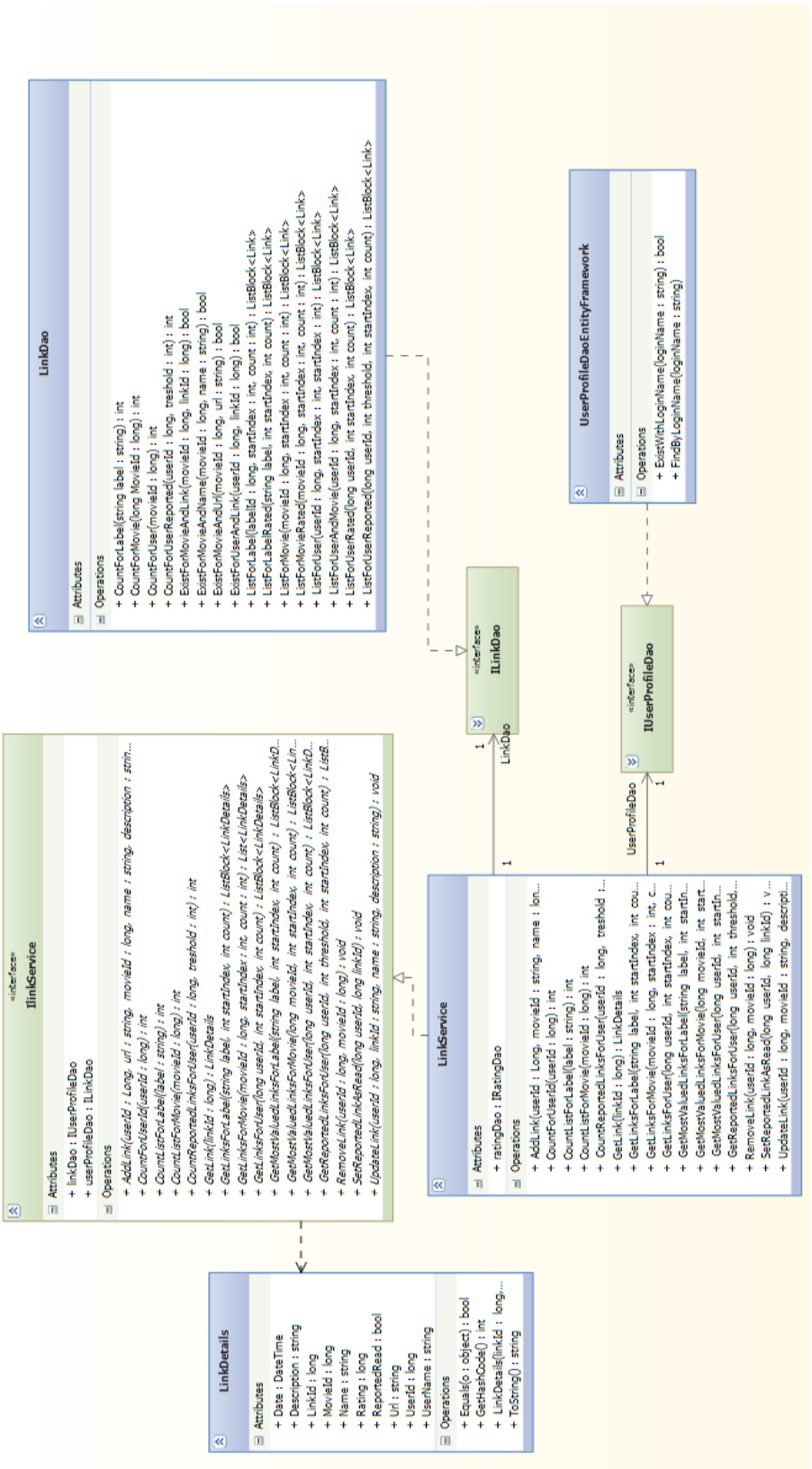

Ilustración 2.11. LinkService

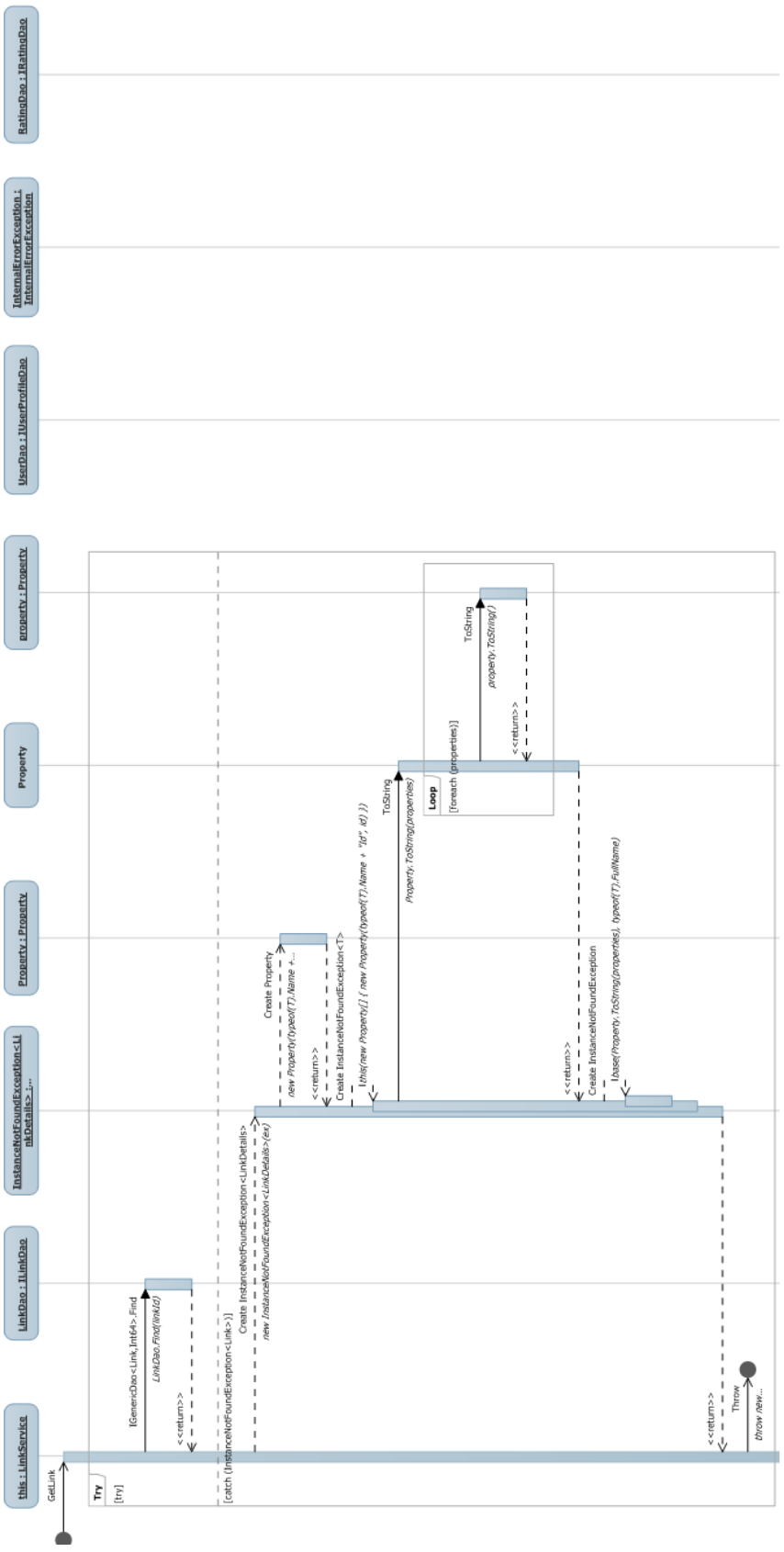

Ilustración 2.12. Método GetLink (1)

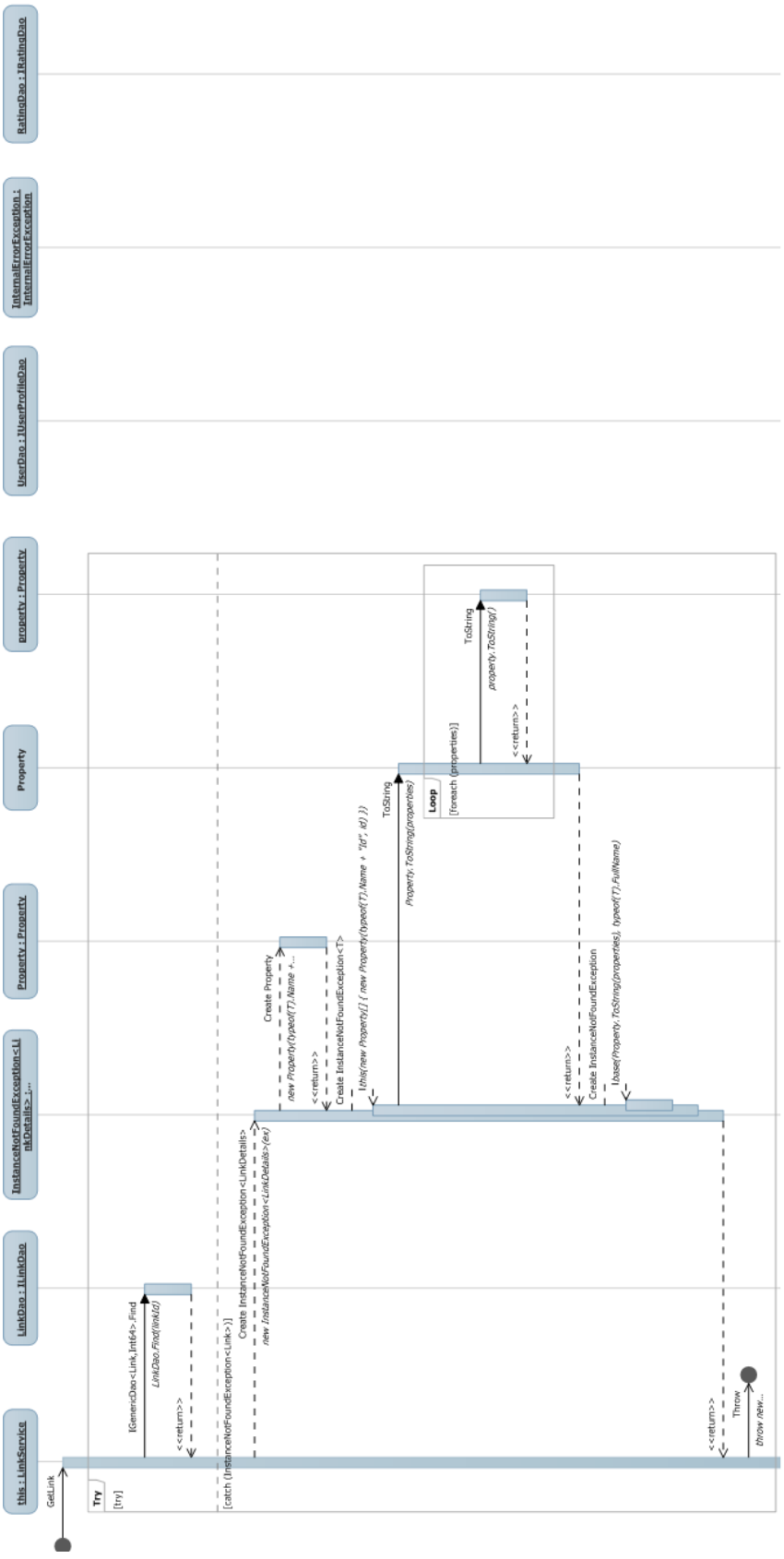

Ilustración 2.13. Método GetLink (y 2)

2.5. Otros aspectos

DAOs y Excepciones. WebMovies/Model/Util

Durante el desarrollo de la práctica se implementaron varias excepciones parecidas en que lo único que cambiaba era el tipo de entidad referida (`NoMoreLinksException`, `NoMoreCommentsException`, `NoMoreFavoritesException`, `UserNotAuthorizedOnLink`, `UserNotAuthorizedOnComment`...). También se detectó que las excepciones que descendían de `InstanceException`, solamente podían referir su instanciación por el identificador, cuando en realidad en diversos casos en que una entidad fallaba a la hora de instanciarse era por alguna otra característica. Para ello, cada excepción encapsula una colección de un nuevo struct Property, par nombre-valor, los cuales se pueden introducir instanciándolas o de forma transparente en los constructores. Por tanto, las nuevas excepciones están parametrizadas según la entidad que refieren y las excepciones de `ModelUtil` se extendieron haciéndolas genéricas y permitiendo indicar qué propiedades con qué valores eran las que se requerían para la instanciación.

Para generar estas excepciones y para mantener la coherencia lógica, se creó un DAO genérico que extiende el de `ModelUtil`, captura las excepciones originales y lanza las nuevas. Adicionalmente en el `GenericDao` no se incluía una excepción cuando se intentaba actualizar una entidad que no existía y se consideró adecuado añadir esta característica. Así todos los DAOs de la práctica lanzan las excepciones extendidas sin codificación adicional en ellos y para los métodos añadidos existen las nuevas funcionalidades.

De este modo, tanto un servicio que utilizase un DAO como cualquier otro sistema que accediese a un servicio seguirían sin percibir alteración en el comportamiento de la capa inferior, pero si se usaban las nuevas características podrían sacarles partido.

Es de especial interés, el efecto en las pruebas, ya que permite controlar que una excepción esperada que originalmente era simplemente, por ejemplo, `InstanceNotFoundException` ahora es posible controlar si se debe a un `UserProfile`, `Link` o `Favorite`, cuando antes no era posible hacerlo de forma automática y permite una trazabilidad más dirigida.

Uso de LINQ

Para justificar el uso de LINQ como lenguaje de consultas, nos apoyamos en las siguientes ventajas que proporciona:

- Extensibilidad: Es una de sus características, ya que fue diseñado para hacer consultas contra cualquier objeto o fuente de datos.
- Bajo acoplamiento con la fuente de datos: El esquema de programación usado en LINQ es consistente, por lo que la sintaxis, es idéntica, no importa contra que fuente hagamos una consulta, ficheros XML, distintas BD con distintos SGBD…
- Verificación de los datos en tiempo de compilación. Intellisense.Aviso de algún cambio por medio de warnings: LINQ ofrece un entorno (Strongly typed) en el desarrollo de aplicaciones, y esto es una ventaja para verificar si algún dato de la fuente de datos se ha modificado en tiempo de compilación.
- Ayuda a resolver el problema de incompatibilidad: Cuando hablamos de ORM, tendremos en cuenta el término (“impedance mismatch"), que se resume a la siguiente expression: Datos != Objetos y mas especificamente Datos Relacionales != Objetos.

Test

En cuanto a las pruebas sobre las listas, comentar que cuando hay una lista comprobamos q no tenga elementos, luego añadimos progresivamente y hacemos pruebas por etapas y comprobamos ordenaciones.

Creación de las clases de utilidades, explicadas en el apartado errores.

Scrispt SQL

En cuanto a la creación de las tablas y relaciones, hemos contemplado algunas ideas para un borrado de usuarios lo más eficiente posible.

Se nos ocurrieron varias ideas que se recogen y están perfectamente documentadas en el script de creación de las tablas. Un resumen de estas soluciones barajadas sería, añadir una columna en el perfil de usuario para mantener un estado de activado o desactivado, de esta manera si un usuario se da de baja en la aplicación, este deja de serlo, pero sigue persistiéndose en la BD, con la idea de que si se mantengan sus comentarios o valoraciones, estas persistan.

Otra opción sería moverlo a otra tabla "alternativa", se recogerían los usuarios eliminados y se migrarían a otra tabla, de manera que podamos diferenciarlos y hacer diferentes operaciones con ellos.

La tercera opción, es fijar los mensajes y ratings su usuario a null si es que este se ha borrado, y es la opción que nos parece más correcta.

Por último, la peor opción de todas, porque sería la menos real o lógica, aunque nosotros la usamos en la práctica, debido a la falta de tiempo, es el que si se borra un usuario se borran sus comentarios y sus ratings.

Además hay creados dos scripts sql uno para la creación de tablas y otro para poblarla con inicializaciones.

3. Interfaz gráfica

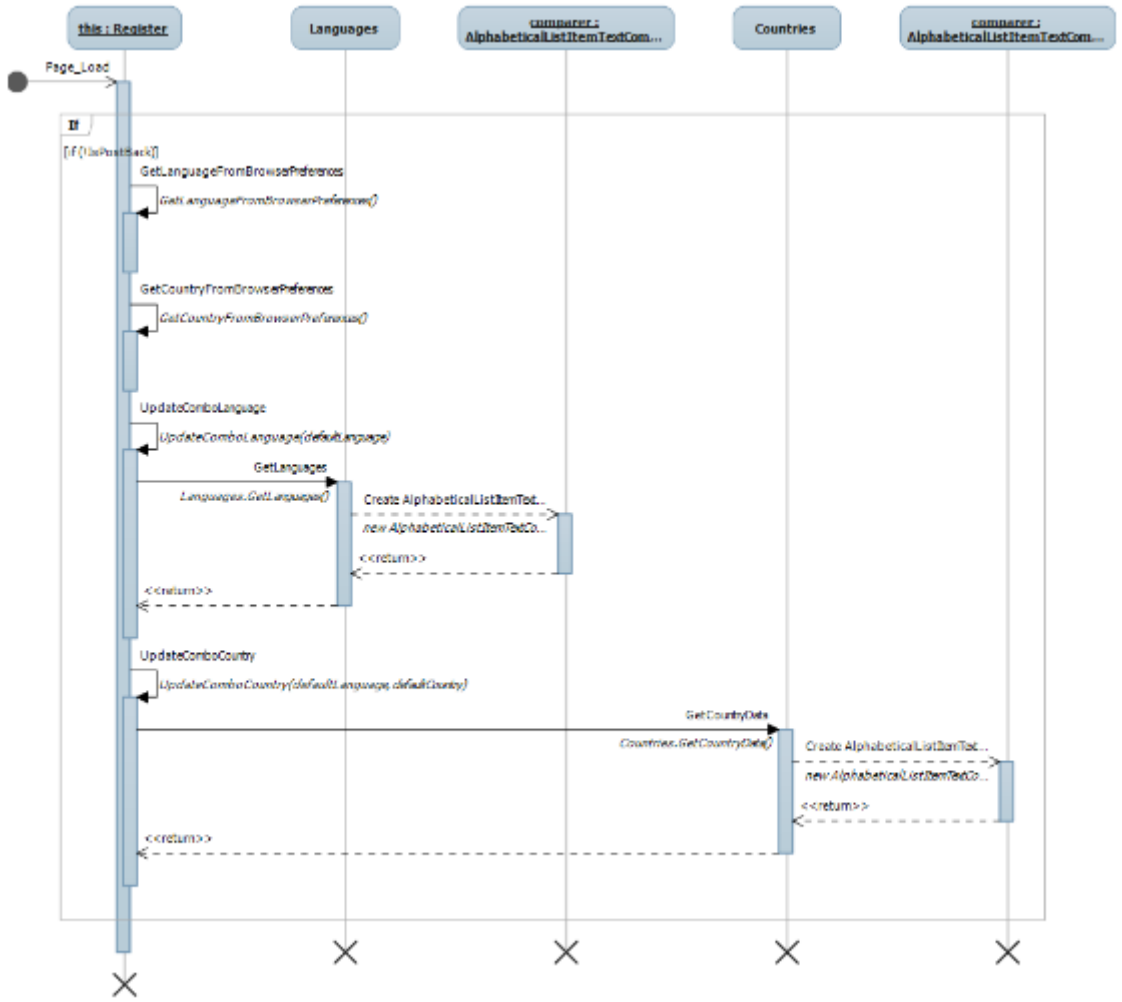

Ilustración 3.1. Register Page_Load

En este apartado, se muestra un diagrama de secuencia, que intenta explicar la interacción entre distintos elementos de la capa web cuando se llama al evento Page_Load, de la página register.

Lo más importante, de esta secuencia se resume en decir que cuando se redirige el control a esta página, si es la primera vez que se visita, es decir no se ha hecho nunca postback, se establece el lenguaje y el país del usuario en esa sesión según la configuración por defecto del navegador.

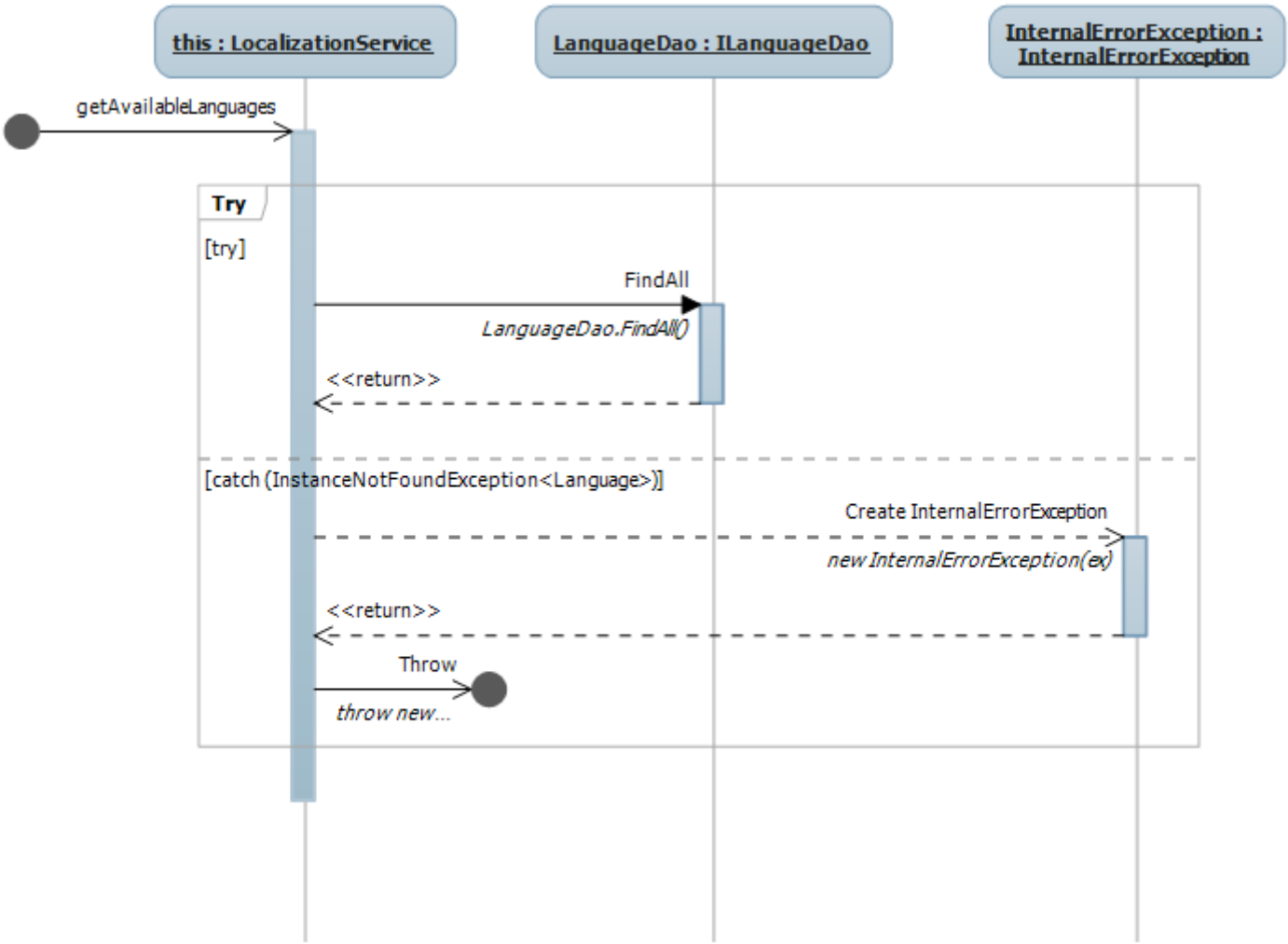

Ilustración 3.2. Recuperación de lenguajes disponibles

A continuación comprueba y actualiza el lenguaje y país, con el que se navegará según lo seleccionado en el comboBox, en la página de registro de usuario. La SpecificCulture de esta página hereda de Page, y su culture se establece según el valor que el usuario introduzca en el combo. Esos eventos se registrarán, y se instanciarán con el idioma marcado por el usuario, y se modificará la lengua y pais de navegación por la páginas, en caso de no haber marcado ninguno, se mantiene en default del navegador.

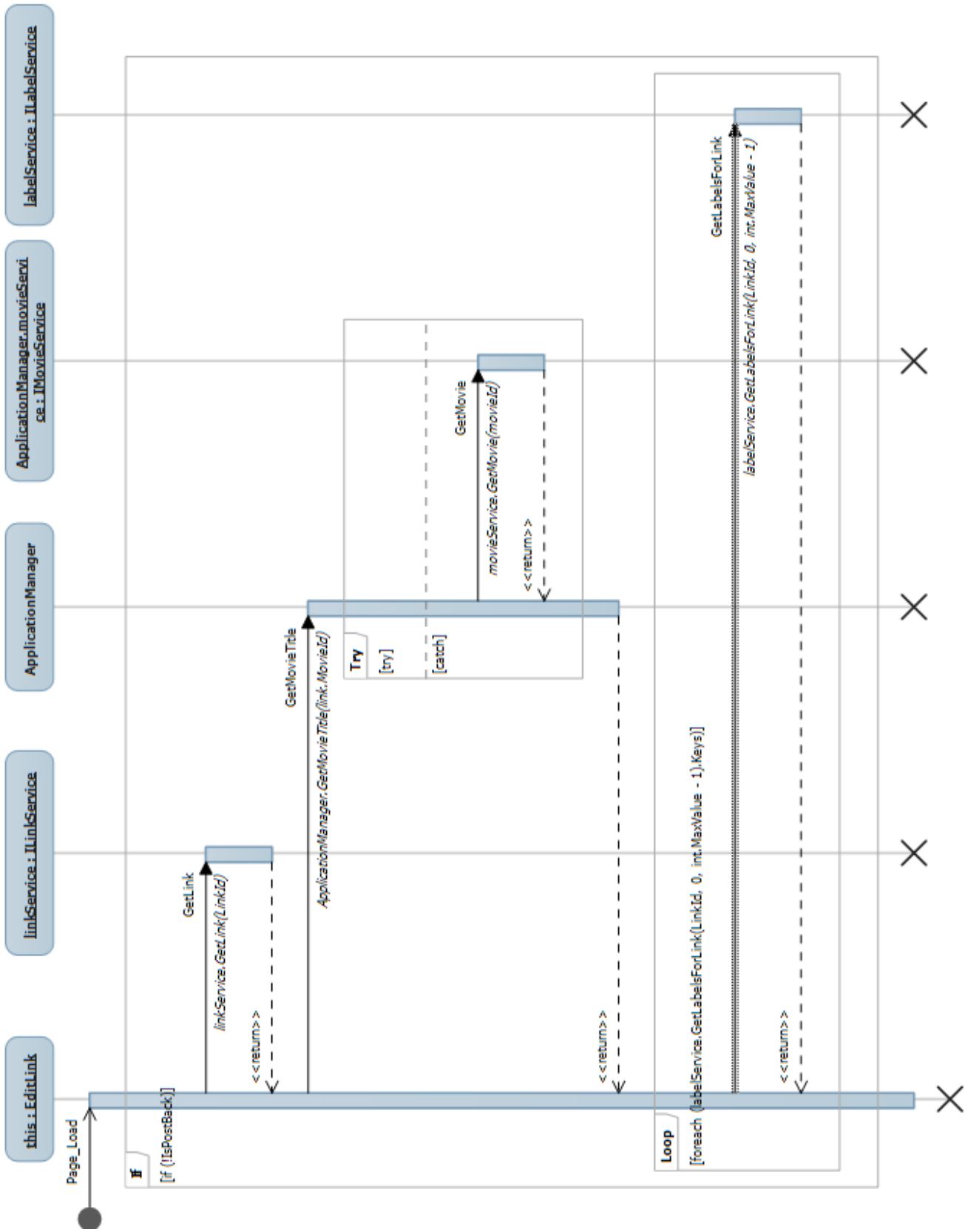
this : EditLink
linkService : ILinkService
ApplicationManager
ApplicationManager.movieServi ce : IMovieService
labelService : ILabelService
Page_Load
If
[if (!IsPostBack)]
GetLink
linkService.GetLink(LinkId)
<<return>>
GetMovieTitle
ApplicationManager.GetMovieTitle(link.MovieId)
Try
[try]
[catch]
GetMovie
movieService.GetMovie(movieId)
<<return>>
<<return>>
Loop
[foreach (labelService.GetLabelsForLink(LinkId, 0, int.MaxValue - 1).Keys)]
GetLabelsForLink
labelService.GetLabelsForLink(LinkId, 0, int.MaxValue - 1)
<<return>>

Ilustración 3.3. EditLink Page_Load

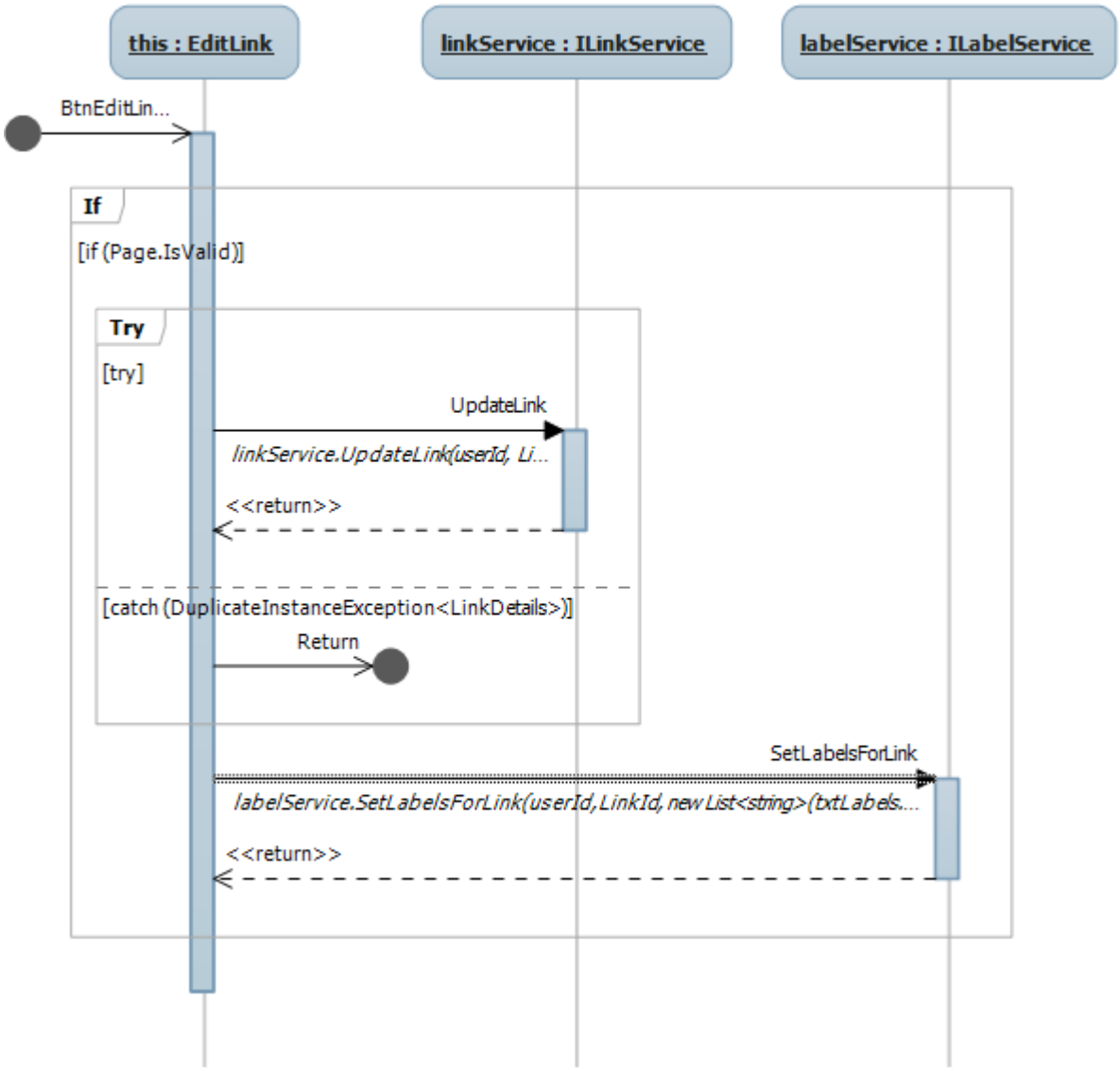

Ilustración 3.4. Botón EditLink

4. Apartados opcionales

4.1. Recomendaciones sobre enlaces

Cuando un enlace alcanza un valor determinado la aplicación lo muestra como promocionado, con un color resaltado y un sombreado especial. Del mismo modo, cuando recibe un número determinado de votos negativos este enlace aparece como denunciado, presentándose sombreado y oscuro.

Cuando un enlace es denunciado a su autor se le añadirá a la lista de enlaces denunciados y aparecerá un marcador con un icono con forma de banderola en su menú, mostrando el número de enlaces que se encuentran por debajo del umbral. Accediendo a él el usuario podrá ver sus enlaces denunciados, comprobar por qué fueron votados negativamente y tomar las acciones adecuadas. Al usuario se le proporciona un nuevo control sobre cada uno de esos enlaces en forma de una banderola, la cual, una vez marcada, representa que el usuario ha aceptado el aviso. Este aviso una vez hecha la revisión dejará de aparecer aunque seguirá pudiendo acceder a sus enlaces desde la vista general de enlaces en su menú.

Los valores umbrales se controlan a nivel de aplicación en el proyecto web, almacenándose éstos valores en las propiedades de aplicación. A la hora de llamar a los servicios se ofrecen 3 nuevos métodos en la fachada de `ILinkService`: `GetReportedLinksForUser()`, `CountReportedLinksForUser()` y `SetReportedLinkAsRead()`. Los dos primero métodos añaden sobre la llamada original un parámetro con el umbral que se quiere emplear para determinar cuáles habrían sido denunciados, ya que este umbral podría cambiar (sería válido uno dinámico en función de varios parámetros).

A nivel de modelo simplemente se almacenará si un usuario ha comprobado que el enlace ha sido revisado una vez denunciado para elaborar la lista, propiedad que se podrá activar con el tercer método indicado, `SetReportedLinkAsRead()`.

4.2. Parsing de XML

Para ofrecer la funcionalidad de parsing de los XML, se ha optado por cambiar los métodos originales del servicio `MovieService` añadiéndoles el sufijo `–Xml`, indicando que el método que devuelven es un documento XML, y hacer que los métodos originales, `GetMovie()` y `SearchMovies()`, devuelvan una clase (o un `ListBlock` de ellas) con los detalles de una película: `MovieDetails`. A nivel de "DAO" (aunque propiamente al acceder a un servicio web externo, no es propiamente un DAO), no serán necesarios cambios en la clase `MovieProxy`, que seguirá proporcionando documentos XML recuperados de ***WebShop*** a partir de la petición construida en función de los parámetros proporcionados.

El servicio intentará recoger el documento XML del proxy al servicio web y si no es posible lanzará una nueva excepción: `NetworkException`, extendida de `InternalErrorException`. Una vez obtenido el documento se navegará a través de él empleando las funcionalidades que ofrece C# y se construirá el objeto `MovieDetails` con los datos proporcionados y se consultará al `LinkDao` el número de enlaces que tiene esa película para incluirlo. Si el documento recuperado contuviese un error, este también se parsearía para proporcionar la respuesta apropiada, en forma de `InstanceNotFoundException<MovieDetails>` o de lista vacía. Así mismo si no fuese posible analizar el XML porque el formato no fuese el esperado se lanzará una `XmlException`, también extendida de `InternalErrorException`, ya que no es un comportamiento esperado.

Para mostrar esta nueva opción a nivel de web, a la página principal se le añade un combo a la barra de búsqueda para seleccionar entre las búsquedas normal y avanzada, las cuales, siguiendo la idea empleada en la fachada del servicio, redirigirán a la página ASP con el sufijo –Xml o sin él, respectivamente.

Las nuevas páginas, Movie.aspx y ListMovies.aspx (frente a las originales renombradas a -Xml), emplean el code behind para rellenar con los datos recuperados del servicio accediendo a las propiedades de los `MovieDetails` recuperados, en los cuales se incluye el conteo de enlaces que será mostrado. En el caso del listado de películas, como ya era norma en los demás listados en web, se emplea un `ListView`.

5. Compilación e instalación

Como paso previo a la ejecución de la aplicación ***WebMovies***, será necesario tener corriendo Tomcat y MySQL sirviendo en el puerto 8080 del equipo local la aplicación Java EE de la primera parte de la asignatura, ***WebShop***, siguiendo las indicaciones que ya se trataron en esa parte.

Toda la compilación y ejecución se realizará desde el Visual Studio, ya que dispone de todas las herramientas y servicios integrados. Si no se dispone de la aplicación se podrá hacer checkout del repositorio SVN en https://svn.fic.udc.es/ei5/is/11-12/isg026/WebMovies/. Se ha incluido ya el fichero `ModelUtil.dll` dentro de la distribución, por lo que no es necesario añadirlo.

La creación de la base de datos en el gestor de SQL Server 2008 Express, incluido en el propio Visual Studio, se hará ejecutando el script `SqlServerInitializeDatabase.sql` contra el servidor `localhost\SQLExpress`. Si deseamos realizar las pruebas de integración adicionalmente podremos ejecutar a continuación SqlServerPopulateForIntegrationTest.sql, aunque deberemos evitarlo si queremos realizar las pruebas de unidad. Esto es debido a que los elementos creados tienen que tener unos identificadores apropiados para las referencias cruzadas. Se podrían hacer unas inicializaciones más complejas pero no es el objetivo de esta práctica.

Para realizar las pruebas de unidad simplemente con ejecutar la acción del menú `Test > Run > All tests in solution` (o el atajo Ctrl+R, A), se nos mostrará el panel de resultados.

Gracias a que Visual Studio integra el ASP.NET Web Development Server, un sencillo servidor ASP.NET para dominio local, la ejecución de la aplicación se puede realizar simplemente ejecutando la acción `View in browser` sobre el proyecto Web, lo cual arrancará una instancia del servidor y abrirá la página con el navegador predeterminado del sistema. La aplicación se despliega en éste mismo, corriendo contra el gestor de bases de datos de SQL Server también integrado. Si quisiésemos desplegar la aplicación en producción, sería necesario configurar un Internet Information Server (IIS) y una base de datos completa (SQL Server, por ejemplo) y usar la opción `Publish...` también sobre el proyecto Web.

6. Problemas conocidos

- La primera vez que se accede a la web se solicita la autenticación en vez de mostrar la página principal. Hemos hecho varias comprobaciones en el fichero `Web.config` prestando especial atención a los tags `<location />` y no hemos conseguido averiguar la razón de este comportamiento.
- A nivel de modelo se permite que un usuario tenga en favoritos un enlace propio, sin embargo a nivel de web se controla que no pueda hacer esto.
- Aunque las pruebas de unidad se han diseñado se forma exhaustiva se ha observado un comportamiento anormal a la hora de eliminar entidades que estén relacionadas con otras. Este comportamiento se puede evitar cargando los elementos usando el método `Includes()` en LINQ. Sin embargo, en caso de que la carga se haga llamando a un método del `GenericDao` provisto en el ModelUtil, los elementos relacionados se cargan correctamente pero a la hora de eliminarlos se lanza una excepción. Desconocemos la causa de este comportamiento y buscamos una alternativa para realizar las pruebas.