Facultad de Informática / Informatika Fakultatea

TITULACIÓN: Ingeniería Informática

Acceso a catálogo de biblioteca desde móviles Android y navegadores mediante servicios web

Alumno/a: D. Luis Tomás García Sánchez

Director/a: D. Alfredo Goñi Sarriguren

Proyecto Fin de Carrera, julio de 2013

Resumen

Este PFC trata la construcción de un sistema de acceso al catálogo de una biblioteca doméstica a través de distintas interfaces de usuario que hacen uso de unos mismos servicios web REST expuestos en un servidor Java EE. Su desarrollo implica la integración de múltiples tecnologías open source a varios niveles.

Keywords: Java EE, REST, Android, Vaadin

Índice

1. Objetivo

Este proyecto nace de una necesidad real: la de gestionar de manera efectiva el catálogo de una abultada biblioteca doméstica. Esa manera efectiva pasa ineludiblemente por un inventariado informatizado de los datos que diferencian los libros. Por tanto, la primera motivación de este proyecto informático la constituye esa necesidad real que le imprime su carácter útil.

De esta necesidad que constituye la primera motivación deriva una segunda motivación nacida de lo que podemos llamar *defecto profesional*. Se trata de imprimir al proyecto un fuerte carácter formativo en el plano técnico, que permita afianzar lo aprendido en los anteriores años de estudio o vida profesional, en algunos casos; e introducir en el aprendizaje de otras materias, en otros.

Esta segunda motivación es la que conforma el verdadero objetivo del presente Proyecto de Final de Carrera (en adelante PFC) ya que, si bien la consecución de unos requerimientos funcionales por medio de la implementación de un sistema informático es de provecho para el usuario final de ese sistema, todo lo aprendido en el transcurso de esa implementación constituye el provecho de aquel que ha implementado el sistema.

De este modo, descartamos soluciones ya existentes que permitan la simple catalogación de la librería para construir un sistema técnicamente más complejo y construido a nuestra medida.

Por tanto, este PFC pretende mostrar el camino que lleva a la consecución de ese sistema de catalogación que permitirá gestionar eficientemente nuestra librería doméstica. Particularmente mostrará las decisiones que se han tenido que tomar para conformar la arquitectura del sistema, cuál es ésta y de qué manera se implementa; cuáles eran las alternativas y por qué han primado las soluciones escogidas; qué particularidades se han presentado en el desarrollo, etc. Especialmente, este PFC pretende mostrar de qué manera se orquestan un gran número de tecnologías utilizadas a la hora de implementar este sistema.

Es, al fin, esta segunda motivación en la que la redacción de la memoria del PFC se centrará principalmente.

2. Descripción del escenario

Nuestra librería doméstica consta de un grueso aproximado de unos 700 volúmenes, con un crecimiento anual estimado de entre 60 y 80 volúmenes. Se trata de libros de diversos temas, no necesariamente conexos, que se encuentran en varios idiomas. En algunos casos, la misma obra puede encontrarse en distintas ediciones. En muy pocos casos existen varias copias de una misma edición. Todos los ejemplares se almacenan en un mismo espacio físico.

Este número de volúmenes, y la imposibilidad de concertar en su espacio de almacenamiento un orden determinado, hace que el inventariado de la biblioteca y, sobre todo, su manejabilidad se haga tediosa; recordar si se tiene un libro en concreto o disponer de todas las obras presentes en la biblioteca sobre determinado tema no son tareas sencillas.

Además, se dispone de un, por el momento, pequeño catálogo de libros electrónicos (*ebook*) de también diverso temario, etc., en diversos formatos. Algunos *ebooks* están disponibles en más de un formato. Existen obras para las que se dispone de ediciones tanto en versión impresa como digital.

A continuación se detallan algunos ejemplos de obras presentes en la biblioteca que ilustran algunas de las casuísticas antes mencionadas:

1. Obra: *Don Quijote de la Mancha*, Miguel de Cervantes

   Ediciones:

   1.1. Edición de Francisco Rico

   Madrid: Punto de lectura, 2007

   1.2. Edición del Instituto Cervantes, dirigida por Francisco Rico

   Navarra: Círculo de Lectores, 2004

   1.3. Primera parte, edición de John Jay Allen

   Barcelona: Cátedra, 2010, 29ª edición

   1.4. Segunda parte, edición de John Jay Allen

   Barcelona: Cátedra, 2009, 28ª edición

   1.5. Edición digital en formatos EPUB y MOBI

2. Obra: *La divina comedia*, Dante Alighieri

   Ediciones:

   2.1. Versión poética de Abilio Echeverría

   Madrid: Alianza, 2004

   2.2. Versión de Joan F. Mira

   Barcelona: Proa, 2004

   Edición bilingüe italiano-catalán

   2.3. Inferno, a cargo de Natalino Sapegno

   Milán: La nuova Italia, 2005

   Edición en italiano

   2.4. Purgatorio, a cargo de Natalino Sapegno

   Milán: La nuova Italia, 2005

   Edición en italiano

   2.5. Paradiso, a cargo de Natalino Sapegno

   Milán: La nuova Italia, 2004

   Edición en italiano

2.1. Requisitos

Se desea que el sistema de inventariado de esta biblioteca doméstica permita, en primer lugar, almacenar la siguiente información referente a los libros:

- Título y autor de la obra
- Nombre de la editorial, año y ciudad de publicación
- Número de la edición, en caso de ser mayor a la 1ª
- ISBN[1] y/o EAN[2]

El ISBN y el EAN sólo se almacenarán si el libro posee alguno de ellos. En caso de tener ambos es deseable que se almacene el EAN, ya que se trata de un identificador más actual.

---

[1] *International Standard Book Number*, Número Estándar Internacional de Libro. Es un identificador único para libros, previsto para uso comercial.

[2] *European Article Number*, Número Europeo de Artículo. Es un sistema de códigos de barras. Desde enero de 2007, el sistema EAN-13 se compatibiliza con el ISBN-13 en España. http://www.mcu.es/libro/CE/AgenciaISBN/InfGeneral/ISBN13.html

Respecto al autor de la obra, es necesario diferenciar la autoría de la obra respecto a cualquier otra relación que tuviera con ella. Así, se desea conocer para un libro el autor de su obra original, el editor que pudiera tener, director, traductores, prologuistas, etc. No se trata de un registro sistemático de todas las personas implicadas en la edición de un libro sino de almacenar esta información relativa a la participación en los casos en los que ésta sea relevante. Por ejemplo, en el caso de las ediciones del *Quijote* que antes exponíamos es muy relevante la información acerca de su editor. El resto de relaciones se han de tratar de un modo similar, en función de la relevancia de la información que proporcionaría tenerla registrada en el sistema.

Hay que tener en cuenta que existen obras para las que no se conoce su autor (como lo es el *Poema del Mio Cid*), y otras que presentan varios autores (como muchos manuales de Historia, por ejemplo).

Del autor se desea conocer su nombre y apellidos. La única información obligatoria ha de ser el nombre, ya que existen autores para los que tan sólo se conoce un seudónimo, pudiendo únicamente informarse el nombre. No es necesario almacenar más información que esta. Es conveniente, eso sí, tener un apartado en el que introducir información varia sobre el autor como un alias o cualquier otra nota que se creyera conveniente. Además, se desea que exista una relación directa entre un autor y sus obras o participaciones en ediciones, de modo que sea fácilmente accesible toda la obra disponible de ese autor.

En lo que a la editorial respecta, hay que tener en cuenta que aunque un libro siempre es editado por una editorial en la mayoría de los casos, o una persona en los pocos casos en los que se trata de una edición personal; puede ser que el libro no presente datos acerca de su edición, por lo que esta información no ha de ser obligatoria a la hora de registrar un libro.

De la editorial se desea almacenar la información referente a su nombre, país y ciudad. En este caso sólo será obligatorio introducir el nombre de la editorial, siendo el resto de información opcional. Del mismo modo que de un autor deseamos conocer su relación completa respecto a los libros registrados en el sistema, de cada editorial registrada en el sistema se desea conocer qué libros están presentes en nuestra biblioteca.

En cuanto a los números de edición, el sistema ha de permitir registrar dos libros pertenecientes a la misma obra y editorial pero de ediciones distintas, de modo que sean distinguibles, por ejemplo, por posibles nuevos colaboradores presentes en la edición más posterior.

Aparte de la información referente a la edición y autoría de las obras, se desea que las obras sean fácilmente categorizables, en función de los temas que se tratan en cada una de ellas. Este sistema de categorización ha de ser sencillo y suficientemente flexible como para añadir nuevas categorías y subcategorías en cualquier momento sin que resulte traumático para las ya existentes.

El catálogo inventariado por el sistema ha de ser explorable gráficamente, preferentemente a través de un navegador web y de forma descentralizada. Esto es, se desea poder utilizar distintos equipos para navegar por el catálogo.

Estos requisitos aplican tanto para los volúmenes físicos como para los electrónicos. La aplicación ha de especificar en qué soporte se encuentra el ejemplar (físico, electrónico/digital, o los dos). No es requisito, pero sí es deseable, facilitar el acceso a la versión digital, si la tuviere, de un volumen.

3. Solución propuesta

Para responder a estas necesidades se propone la construcción de un sistema que permita al usuario: 1) registrar, consultar, modificar y borrar la información expuesta en los requisitos; 2) gestionar un sistema de etiquetas que proporcione un método de categorización y meta-información a los volúmenes; y 3) una herramienta de gestión de la biblioteca ágil.

Este sistema será accesible a través de un navegador web y desde una aplicación para dispositivos móviles Android.

3.1. Plano funcional

A través del navegador web podremos acceder, para cada una de las entidades identificadas en los requisitos (Autor, Obra, Edición y Editorial), a un formulario de creación (y posteriormente edición) y a una serie de relaciones. Así, por ejemplo, para un Autor dispondremos de su formulario de edición de datos y tablas con las obras que tiene relacionadas, bien por ser el autor directo de ellas o bien por haber participado de algún modo en ellas.

Las entidades Obra y Edición se refieren a la obra original, en el caso de la primera; y a la edición en concreto de la obra, en el caso de la segunda. De este modo, a los libros más comunes que encontramos en nuestra biblioteca les corresponden una obra y una edición, mientras que a otros (como los expuestos en el apartado de Descripción del escenario) les corresponden una obra y múltiples ediciones.

La interfaz suministrada para navegadores permitirá así el registro, consulta, modificación y borrado de autores, obras, ediciones y editoriales, así como el acceso a la información derivada de sus relaciones.

Por otra parte, la aplicación para móviles Android permitirá sólo las labores de consulta sobre los datos almacenados de esta biblioteca.

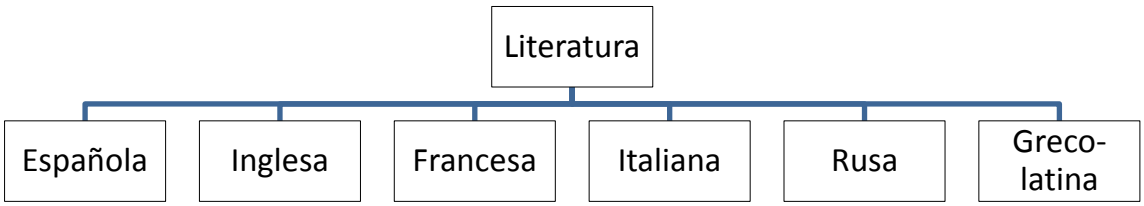
Como añadidos se incluirán imágenes para ilustrar a los autores y las ediciones (utilizando por ejemplo sus portadas). En el caso de las ediciones se proporcionará un enlace para descargarla si se trata de una edición digital de la obra.

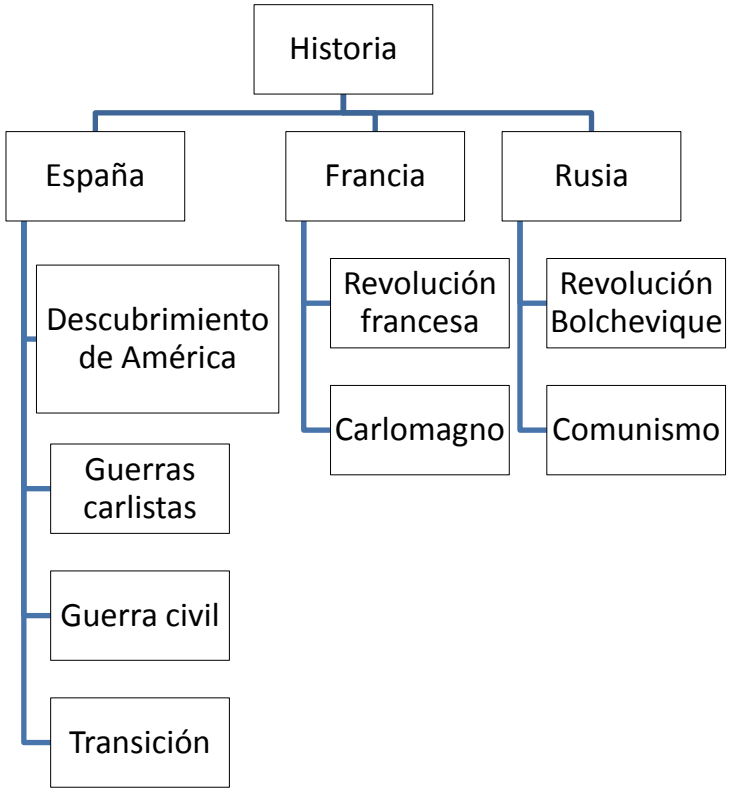
En lo que respecta al sistema de categorización de la biblioteca, éste consistirá en la asignación de etiquetas a las obras en función de criterios diversos. Una obra podrá contar con varias etiquetas.

Estas etiquetas pertenecen a su vez a una categoría, de modo que posteriormente podremos definir nuevas categorías para nuevas etiquetas. Inicialmente se proponen tres categorías de etiquetas: Género, Época y Tema. Las etiquetas pueden tener etiquetas hijas, ordenándose jerárquicamente en forma de árbol, de modo que éstas vayan de lo más

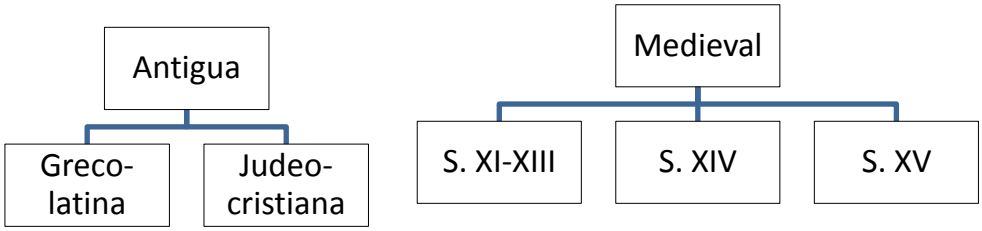
general a lo más específico, haciendo el recorrido desde las etiquetas padre a las etiquetas hijas. Por ejemplo:

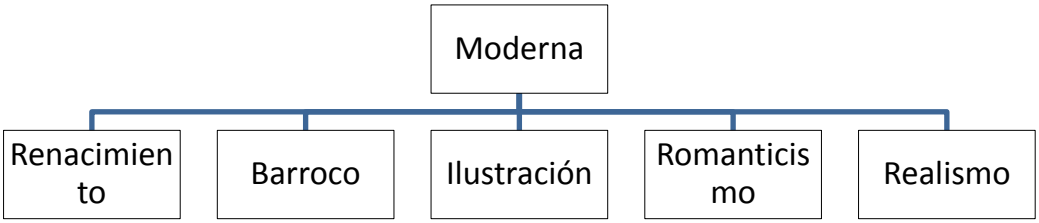
Tema





Época





En los ejemplos arriba expuestos vemos dos categorías distintas: 'tema' y 'época'. En el caso de la categoría 'época', encontramos tres etiquetas de primer nivel: 'antigua', 'medieval' y 'moderna'. Cada una de estas etiquetas presentan unas etiquetas hijas que añaden un grado mayor de especificidad.

A la hora de etiquetar una obra, ésta recibirá toda la información que proporcionen las etiquetas asignadas y sus etiquetas padre. Por ejemplo, a la hora de etiquetar *El Quijote* con los ejemplos expuestos utilizaríamos dos etiquetas: Tema/Literatura/Española y Época/Moderna/Barroco.

Si más tarde exploramos las obras etiquetadas con Tema/Literatura/Española o Época/Moderna/Barroco, la obra debería aparecer listada entre las etiquetadas; pero también debería aparecer listada si realizamos la misma consulta para las etiquetadas con Tema/Literatura o Época/Moderna.

La gestión del sistema de etiquetas permitirá la creación, modificación, consulta y borrado tanto de categorías como de etiquetas, así como la navegación por ellas.

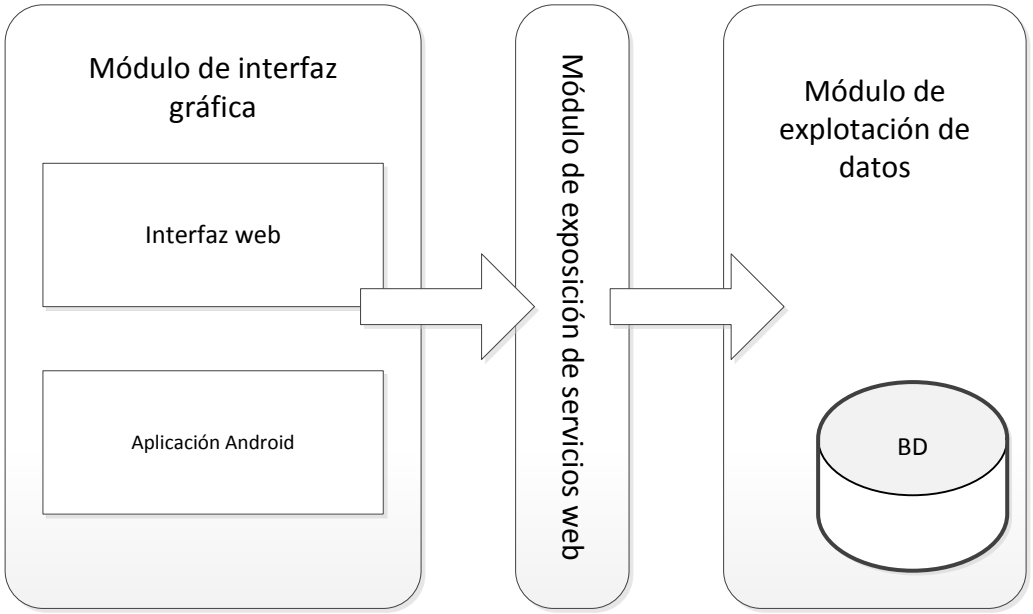
Ambas interfaces de usuario serán de fácil manejo, proporcionando un uso intuitivo de sus funcionalidades.

3.2. Plano técnico

Para implementar la solución propuesta se construirá un sistema modular. El primer módulo estará destinado a la explotación de los datos, ocupándose de su almacenamiento en memoria y su posterior recuperación y actualización.

El segundo módulo expondrá unos servicios web. De este modo, aquellas aplicaciones que consuman estos servicios podrán operar con los datos que tenemos almacenados.

El tercer módulo es el que proveerá las interfaces gráficas para el usuario. Este módulo está a su vez dividido en dos partes, una para la interfaz web y otra para la aplicación Android. Ambas serán consumidoras de los servicios expuestos por el segundo módulo.


Módulo de interfaz gráfica
Interfaz web
Aplicación Android
Módulo de exposición de servicios web
Módulo de explotación de datos
BD

4. Alcance del proyecto

De acuerdo con el planteamiento expuesto en el primer apartado de esta memoria, que trataba acerca del objetivo de este proyecto, se implantará tan sólo una parte de este sistema. Sin embargo, se trata de una parte representativa del sistema completo. Esto es, una parte que abarca todos aquellos temas referentes a la arquitectura que presentaría la implementación completa del sistema.

En concreto, se implementará la relación existente entre un autor y una obra escrita por él mismo. Esta implementación requerirá el uso de todas las tecnologías presentes en la implementación completa del sistema. Dicho de otro modo: la implementación completa del sistema no es más que una ampliación de la parte que cubre este proyecto, sin utilizar nuevas tecnologías ni *frameworks*.

Así, en la redacción de esta memoria podremos pasar por todos los estadios de implementación del sistema ilustrando cada caso con la parte del sistema completo implementada.

5. Análisis de requisitos

A continuación detallaremos el diagrama de clases que dará forma al sistema y estudiaremos los casos de uso implicados en el alcance del proyecto.

5.1. Diagrama de clases

Antes de plasmar el diagrama de clases, desgranaremos de los requisitos expuestos anteriormente las diferentes clases que han de estar presentes en el sistema y las relaciones entre ellas (aunque en el apartado que trataba el plano funcional de la propuesta ya adelantábamos algunas de las entidades que estarán presentes en el sistema). Más adelante trataremos las propiedades de estas clases.

Primeramente, consideraremos el elemento que justifica la construcción del sistema, que es el libro. Describiremos el concepto de libro a efectos de la información que pueda ser relevante para nuestro sistema de catalogación. Así, un libro es el resultado de **editar** una **obra**, escrita por un **autor**. Esta edición la lleva a cabo una **editorial**.

Por recuperar el ejemplo del *Quijote*: Miguel de Cervantes escribe en 1605 su obra *El ingenioso hidalgo Don Quijote de la Mancha*. Esta obra recibe una edición primigenia en el momento de ver la luz, cuando es impresa por primera vez por Juan de la Cuesta. En los años (y siglos) sucesivos, la misma obra se ha ido editando de nuevo de la mano de muchas y distintas editoriales. En este ejemplo vemos cómo una misma obra, perteneciente a un único autor, posee múltiples ediciones de distintas editoriales.

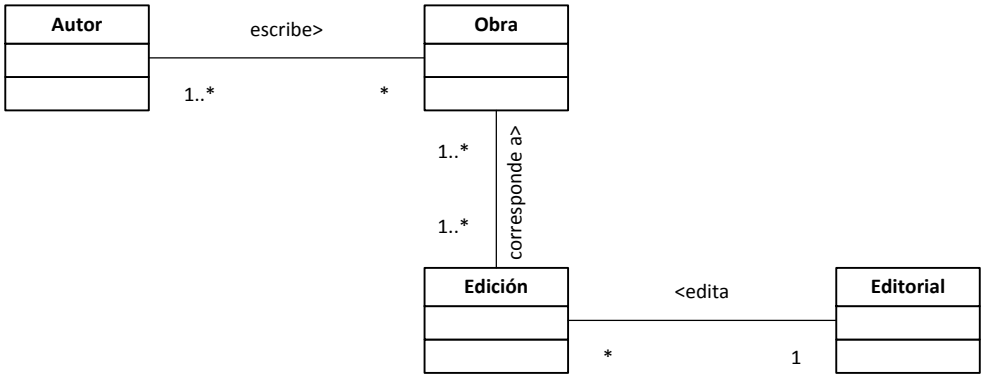
Así pues, en nuestro sistema vamos a aplicar las siguientes restricciones en cuanto a las relaciones de estas cuatro entidades:

- Una **obra** está escrita por **al menos un autor**, pudiendo estar escrita por varios.
- Un **autor** puede tener **varias obras** escritas, pudiendo no tener ninguna escrita si participa en el sistema con otro rol (como editor, como veremos más adelante).
- Una **obra** tiene **al menos una edición** (si no, no la tendríamos registrada), pudiendo tener varias.
- Una **edición** corresponde **al menos a una obra**, pudiendo corresponder a varias (sería el caso de libros que recopilan varias obras).
- Una **edición** está editada por **una editorial**.
- Una **editorial** puede tener asociadas **varias ediciones**.

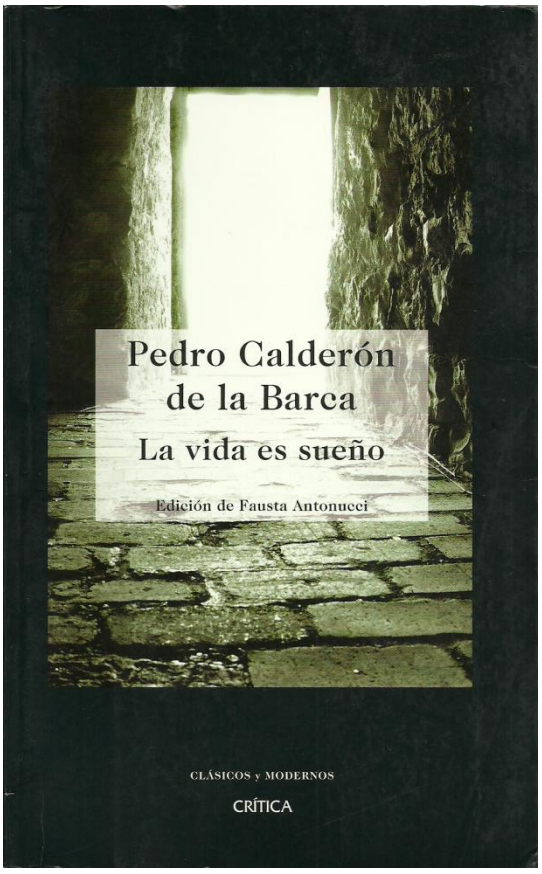
En los requisitos se indicaba que puede ser que de una obra no se conozca el autor (se ponía como ejemplo el *Poema del Mío Cid*). También que de una edición no se conozca la editorial. Esto contrasta con los puntos que acabamos de exponer, en los que restringimos el número de autores de una obra a uno o varios. Es decir, al menos uno. Se debe a que en nuestro sistema existirá un autor 'Anónimo', teniendo éste asociadas todas las obras para las que no se conoce el autor (por ejemplo, el *Poema del Mío Cid* antes mencionado, o el *Lazarillo*).

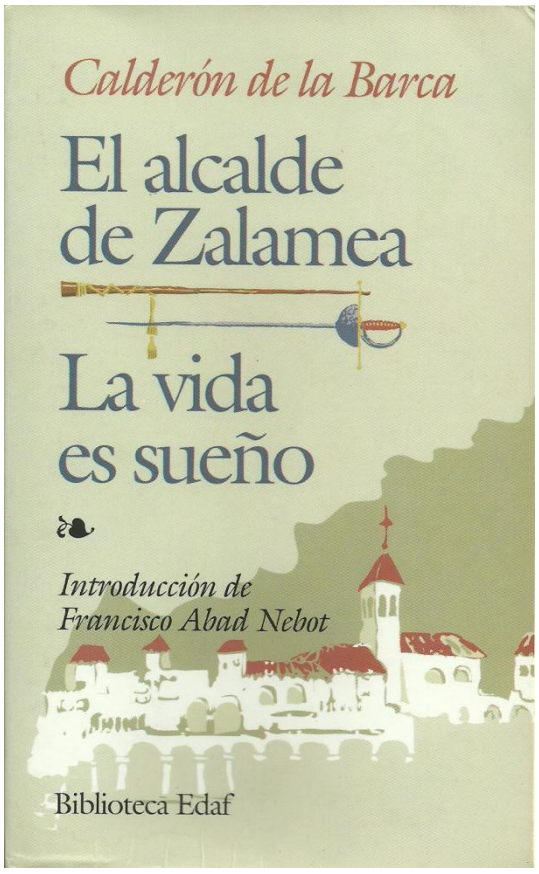
En el caso de la relación entre edición y editorial, el funcionamiento es similar. Hemos indicado que toda edición pertenece a una única editorial. El sistema contará con una editorial 'Desconocida' que tendrá asociadas todas las ediciones para las que no se conoce la editorial original.

Veámoslo gráficamente:



Es fundamental que comprendamos bien estas relaciones. Para ello, pongamos un ejemplo extraído de nuestra biblioteca. Tomemos los dos siguientes libros:



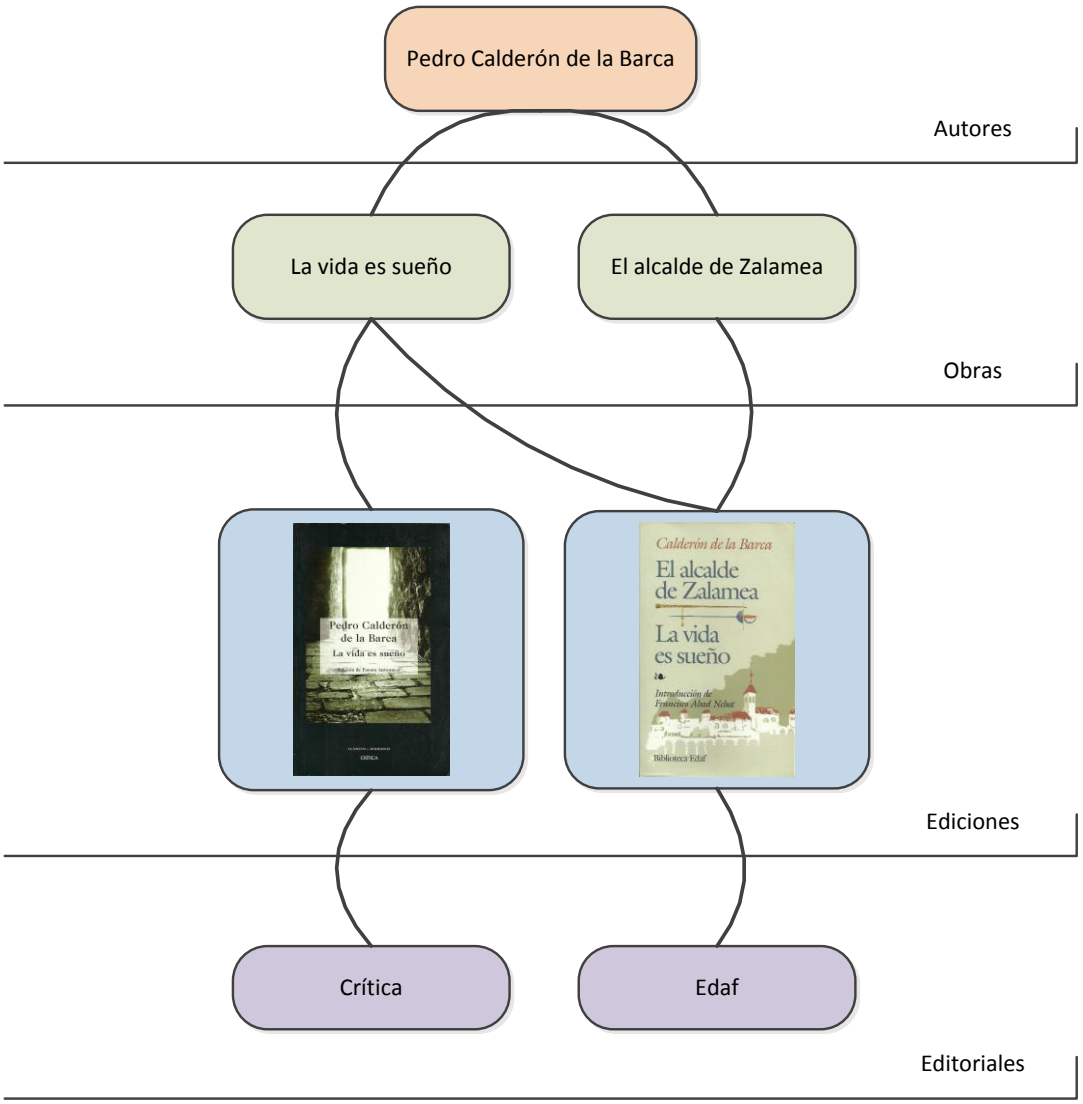


Ambos libros son ediciones de la obra *La vida es sueño*, de Pedro Calderón de la Barca, con la particularidad de que el segundo también contiene la obra *El alcalde de Zalamea*, también de Calderón. De estos libros extraemos la siguiente información para nuestro sistema:

1. **Edición**: La vida es sueño
   Editorial: Crítica
   Obra: La vida es sueño
   Autor: Pedro Calderón de la Barca

2. **Edición**: El alcalde de Zalamea – La vida es sueño
   Editorial: Edaf
   Obras: El alcalde de Zalamea, La vida es sueño
   Autor: Pedro Calderón de la Barca

Veámoslo gráficamente, donde apreciaremos mejor las clases y las relaciones:



En el gráfico podemos observar más claramente cómo un libro (definido esta vez como el objeto físico que almacenamos en nuestras estanterías) se identifica con la clase Edición. De la información que hemos extraído de ambos libros han surgido dos obras, un autor y dos editoriales.

Podemos ver también cómo una misma obra puede tener asociadas más de una edición; y, viceversa, cómo una edición puede tener asociadas más de una obra.

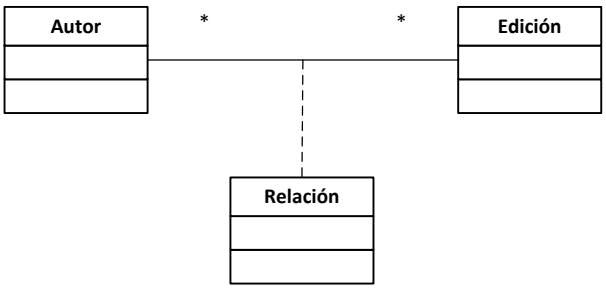
Pasemos ahora al punto de los requisitos en el que se especifica que se desea registrar aquella información que resulte relevante en cuanto a la participación de personas que no sean los autores de la obra original en una edición concreta de esta. En los requisitos se ponía de ejemplo la misma obra del *Quijote* para la cual existían dos ediciones distintas con editores distintos: Francisco Rico y John Jay Allen.

No sólo la participación en una edición como 'editor' puede que sea deseable conocer, sino también la confección de un prólogo a la obra, o la traducción, o la revisión técnica, etc. La persona involucrada en esta participación la identificaremos con la clase Autor que ya tenemos identificada de antes, ya que una misma persona puede tener obras propias y, a su vez, participar en ajenas. Ejemplo de esto último es el mismo Francisco Rico (por seguir con el ejemplo), que además de editor de la obra de Cervantes es autor de la obra *Mil años de poesía española*.

Así pues, en nuestro sistema aplicaremos la siguiente restricción de relación en lo que a participaciones se refiere:

- Un **autor** puede participar en **varias ediciones**, desempeñando **para cada una** de las participaciones **un determinado papel o rol** (que llamaremos **relación**).
- Una **edición** puede tener **varios autores**, **cada uno con un tipo de rol** (o **relación** respecto a la edición) distinta.

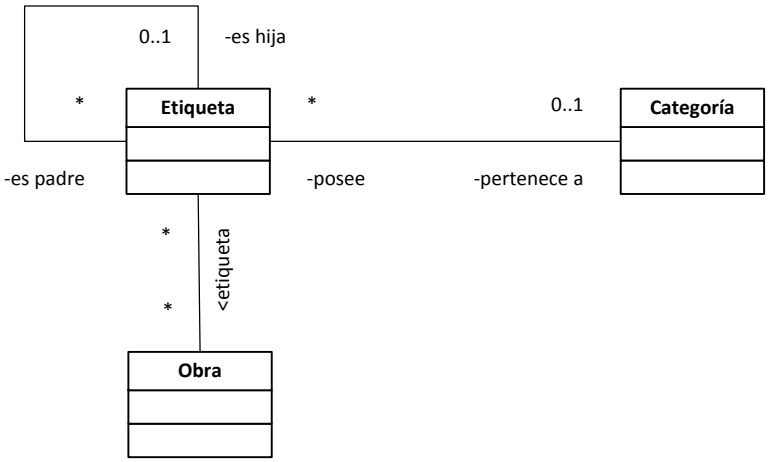
De nuevo, lo vemos gráficamente:



Veamos por último las relaciones correspondientes al sistema de etiquetado de obras. En el apartado de propuesta de solución, comentábamos que este sistema de etiquetado consistiría en la jerarquización de unas etiquetas agrupadas por categorías. Estas etiquetas serán las que se asignen a las obras. Tenemos pues que:

- Una **etiqueta** sólo puede tener **una etiqueta padre**, y puede **no tener ninguna** si se trata de la **etiqueta raíz**.
- Una **etiqueta** puede tener **varias etiquetas hijas**.
- Una **etiqueta raíz** pertenece a una **única categoría**. Una etiqueta que no sea raíz **heredará la categoría de su etiqueta padre**.
- Una **obra** puede tener asignadas **varias etiquetas**.
- Una **etiqueta** puede etiquetar a **varias obras**.

Veamos la parte del diagrama que le corresponde a lo anteriormente citado:



Con las relaciones correspondientes al sistema de etiquetado terminamos el estudio de las clases del sistema y sus relaciones. A continuación detallaremos los atributos de cada una de las clases.

5.1.1. Obra

Para la obra almacenaremos la información referente al título. Esta será la única información obligatoria para una obra. Además, incluiremos un atributo de meta-información para introducir información varia de la obra, a modo de notas sobre la misma. Un tercer atributo, que llamaremos typeSettings, nos hará las veces de atributo auxiliar.

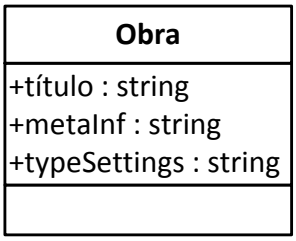
Este atributo auxiliar lo encontraremos también en el resto de las clases. Su cometido es el de proporcionar un recurso en caso de cambio del modelo o lógica en el futuro que precise almacenar información estructurada en la clase. Así, este atributo typeSettings puede contener entradas en forma clave-valor que podrían ser leídas por la lógica del sistema sin ser necesaria una refactorización completa de la aplicación.

Por ejemplo, imaginemos que para todas las obras queremos, en un momento dado del futuro, conocer los años de primera publicación y el idioma original en el que se escribió. En lugar de alterar el esquema de base de datos, refactorizar el modelo del dominio y los servicios, bastará con almacenar en el atributo typeSettings la siguiente información:

original-publish-date=1605,original-language=castellano

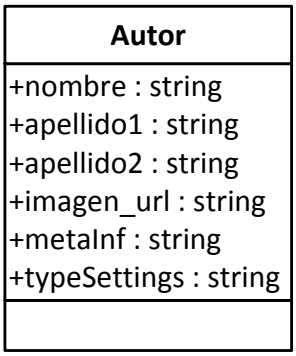
y programar los servicios de tal modo que consulten este atributo. El ejemplo corresponde, cómo no, al *Quijote*, y el nombre de las claves es puramente ilustrativo.

Volviendo a la clase Obra, queda del siguiente modo, con todos sus atributos siendo cadenas de caracteres:



5.1.2. Autor

Respecto al autor, almacenaremos su nombre, primer apellido y segundo apellido; siendo el nombre el único atributo obligatorio. Otro atributo nos permitirá almacenar una URL con una foto del autor. Al igual que para la obra encontramos un campo de meta información y el atributo typeSettings. Todos los atributos son cadenas de caracteres.



5.1.3. Edición

En el caso de la edición, obligatoriamente deberemos informar del título de la edición. Notemos que no necesariamente ha de coincidir con el de su obra original, ya que puede tratarse de una edición que tenemos en un idioma diferente al castellano, o tratarse de una recopilación de obras. Aparte del título, añadiremos atributos para el subtítulo, el número de edición, el año de edición, idioma, ISBN, una URL a una imagen (como, por ejemplo, la portada), la URL que dé acceso al formato digital (ebook) de la edición, y los campos de meta información y typeSettings. Todos los atributos serán cadenas de caracteres salvo los correspondientes al número y año de edición, que serán enteros.

Edición
+titulo : string +subtitulo : string +número : int +año : int +idioma : string +isbn : string +imagen_url : string +ebook_url : string +metaInf : string +typeSettings : string

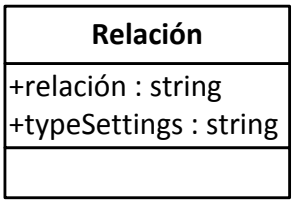
5.1.4. Editorial

Para la editorial el único atributo obligatorio será, como en el resto de los casos, el nombre. El resto de atributos son para la ciudad y el país, además de los campos de meta información y typeSettings. Todos son cadenas de caracteres.

Editorial
+nombre : string +ciudad : string +país : string +metaInf : string +typeSettings : string

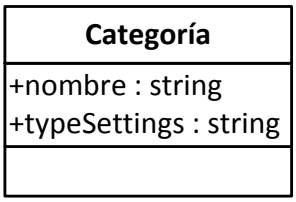
5.1.5. Relación

En lo que a esta clase respecta, tan sólo deberemos asignarle un atributo correspondiente al tipo de relación que representa (si corresponde al editor, al escritor del prólogo, etc.). Se trata de un atributo de tipo cadena de caracteres. Incluiremos también el atributo typeSettings.



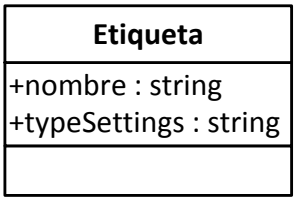
5.1.6. Categoría

La categoría presentará tan sólo un atributo para el nombre y el atributo correspondiente a typeSettings. Todos los atributos de tipo cadena de caracteres.

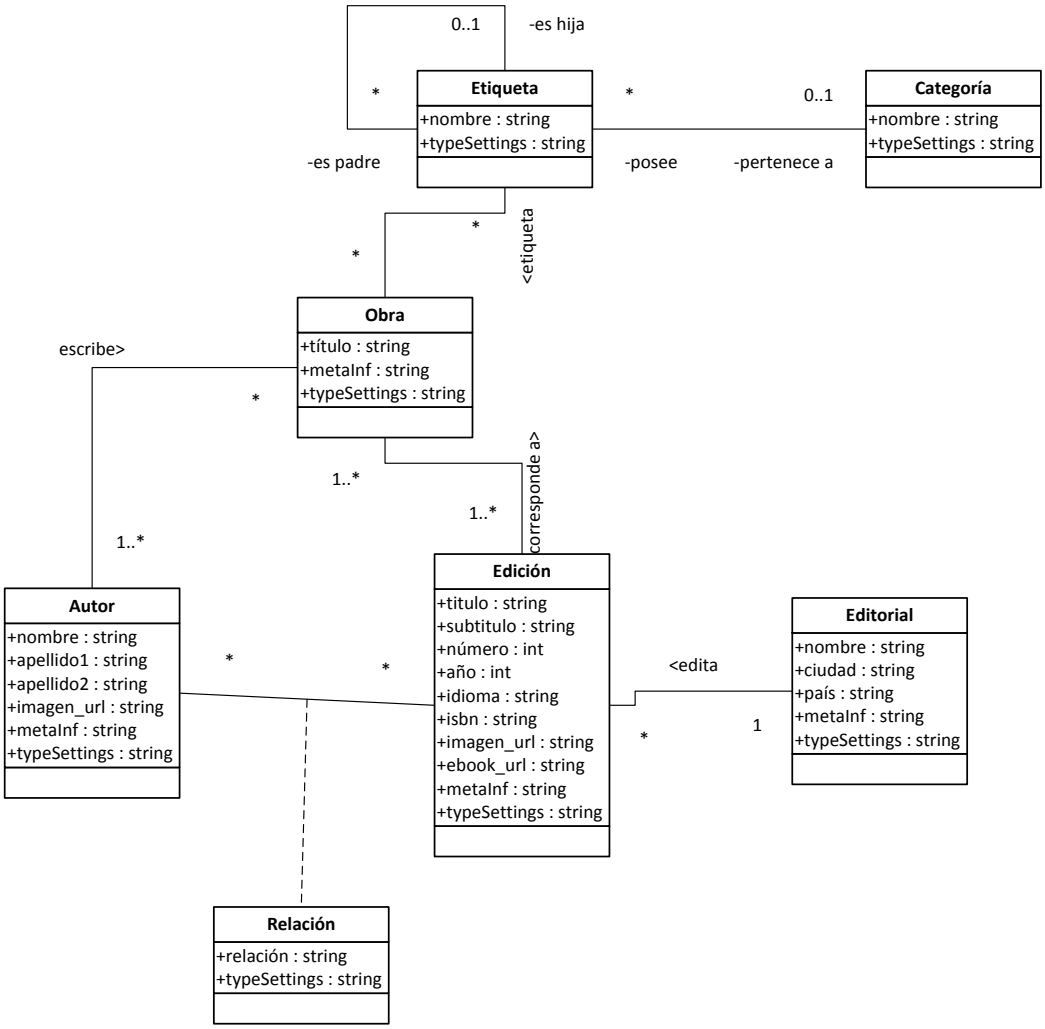


5.1.7. Etiqueta

La etiqueta, al igual que la categoría, presenta dos únicos atributos: el nombre del atributo y typeSettings. De nuevo, del tipo cadena de caracteres.



Después de haber definido las clases, las relaciones entre ellas y los atributos que presentan, es hora de mostrar el diagrama completo de clases a partir de los fragmentos que hemos ido desgranando por el camino.

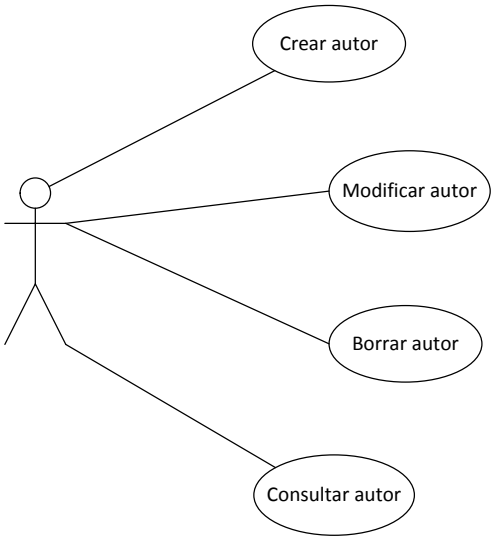


5.2. Casos de Uso

De acuerdo con lo expuesto en el alcance del proyecto, en este apartado veremos tan sólo los casos de uso implicados en la implementación del sistema que desarrollaremos en este proyecto. Es decir, los casos de uso relacionados con las labores de gestión de los autores, las obras y la relación entre estas dos entidades.

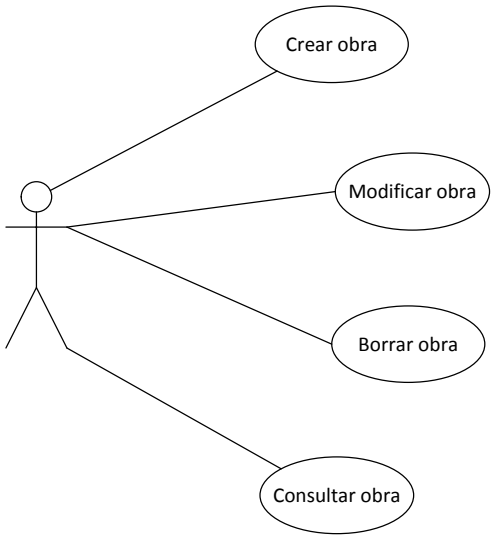
5.2.1. Diagrama de casos de uso

Respecto a los autores, deseamos crearlos en el sistema, modificarlos a posteriori, borrarlos si así lo deseamos y, por supuesto, consultarlos una vez creados.

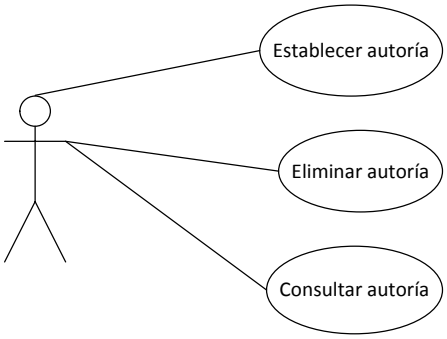


En el diagrama no hemos especificado el actor involucrado en el caso de uso porque en el sistema sólo existirá un rol de usuario, debido a que se trata de un sistema de uso doméstico.

En cuanto a las obras, las operaciones son similares. Debemos poder crear una obra, modificarla, borrarla y consultarla una vez esté introducida en el sistema.



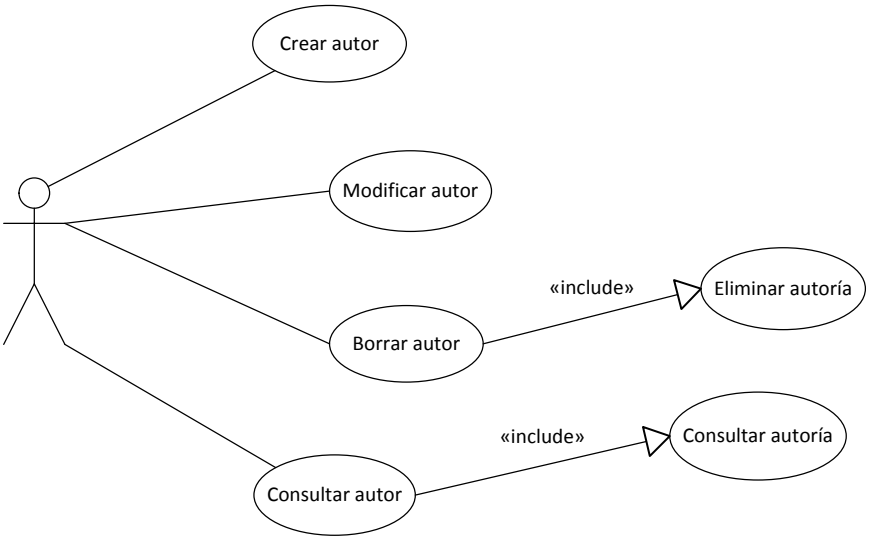
Abordemos ahora la relación entre los autores y sus obras. En este caso, el sistema ha de permitir establecer la relación de autoría entre el autor y su obra, y eliminarla si es preciso. Obviamente, también la hemos de poder consultar.



Esta funcionalidad la integraremos en los procesos de creación, modificación, borrado y consulta de obras y autores para simplificar el sistema. De este modo, al modificar una obra, la relación con sus autores será como un parámetro más de la obra que puede ser modificado. En el caso del borrado de obras o autores, la eliminación de la autoría se llevará a cabo junto al borrado de la obra y/o del autor.

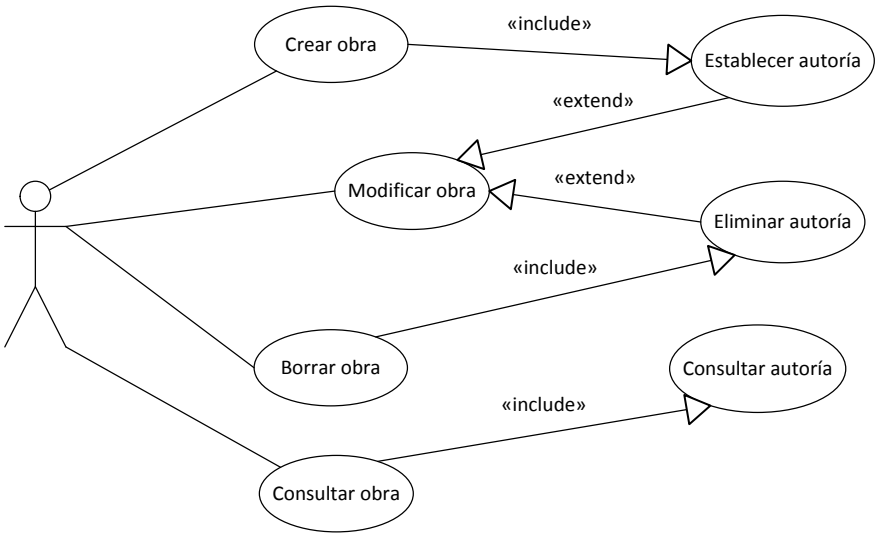
En el caso de la consulta, la autoría se mostrará junto al resto de atributos de los autores y de las obras.

Veamos primero cómo queda el diagrama correspondiente a los casos de uso relativos a los autores:



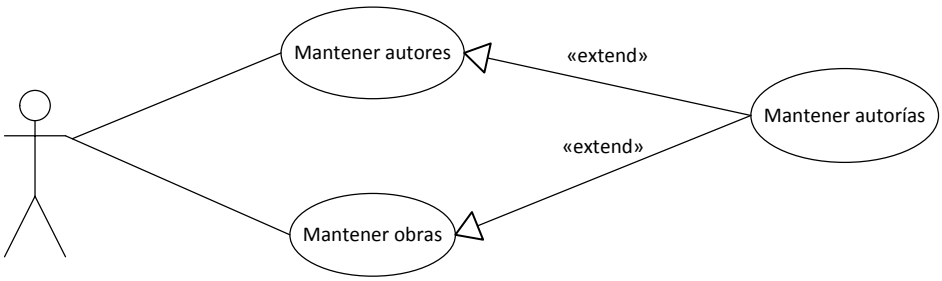
Aquí vemos cómo los casos de uso correspondientes al borrado y a la consulta usan los respectivos a la autoría.

En el caso de las obras, el diagrama queda como sigue:



Como en el anterior caso, el borrado y la consulta de obras incluyen sus correspondientes casos de uso relativos a las autorías. La creación de obras incluye el establecimiento de una autoría, ya que, como hemos visto antes, toda obra ha de pertenecer a, como mínimo, un autor. La modificación, por el contrario, resulta extendida por el establecimiento o la eliminación de autorías en función de si el usuario a la hora de modificar una obra establece o elimina autorías de la misma.

Por simplificar el diagrama final de casos de uso, las operaciones CRUD[3] que hemos contemplado las aglutinaremos en un solo caso de uso de nombre "Mantener". De este modo, todos los casos de uso antes referidos formarán el siguiente diagrama simplificado para la implementación que llevaremos a cabo en este proyecto del sistema:



5.2.2. Representación textual de los casos de uso

Pasaremos ahora a la representación textual de estos casos de uso, de modo que podamos detallar las acciones que se llevarán a cabo en cada uno de ellos. El actor principal recibe el nombre de "usuario" y el sistema recibe el nombre de "biblioteca".

[3] *Create Read Update Delete*, Crear Leer Actualizar Borrar

5.2.2.1. Mantenimiento de autores

Comenzaremos por las operaciones CRUD relativas al mantenimiento de autores en el sistema. Los casos de uso relativos al mantenimiento de las autorías se detallarán en último lugar.

5.2.2.1.1. Crear autor

La creación de un autor consiste en la introducción de los datos relativos a un autor. El sistema se encargará de crear un autor con los datos introducidos.

Caso de uso	**Crear autor**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	-	
Operaciones	Usuario	Sistema
1	Introduce los datos del autor	
2		Registra un autor nuevo con los datos proporcionados

5.2.2.1.2. Modificar autor

La modificación del autor consiste en la modificación de los datos existentes del autor. El sistema proporcionará los datos existentes, el usuario los modificará y, finalmente, el sistema guardará los cambios.

Caso de uso	**Modificar autor**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	El autor existe	
Operaciones	Usuario	Sistema
1		Lista autores registrados
2	Escoge uno para modificar	
3		Muestra los datos del autor
4	Modifica los datos oportunos	
5		Guarda los datos modificados para el autor

5.2.2.1.3. Borrar autor

En el caso del borrado, el sistema eliminará el autor que seleccione el usuario.

Caso de uso	**Borrar autor**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	El autor existe	
Operaciones	Usuario	Sistema
1		Lista autores registrados
2	Escoge uno para borrar	
3		Borra el autor seleccionado
Extensiones		
3.A		Si existen autorías para el autor a borrar, las borra

5.2.2.1.4. Consultar autor

La consulta de los autores consiste en la muestra, por parte del sistema, de los datos de un autor seleccionado, así como de las obras para las que figura como autor.

Caso de uso	**Consultar autor**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	El autor existe	
Operaciones	Usuario	Sistema
1		Lista autores registrados
2	Escoge uno para consultar	
3		Muestra los datos del autor
Extensiones		
3.A		Si existen autorías para el autor, muestra el título de las obras relacionadas

5.2.2.2. Mantenimiento de obras

A continuación detallamos las operaciones CRUD relativas al mantenimiento de obras en el sistema.

5.2.2.2.1. Crear obra

La creación de una obra se lleva a cabo mediante la especificación de los datos de la obra y los autores de la misma. El sistema creará la obra introducida y establecerá las nuevas autorías correspondientes.

Caso de uso	**Crear obra**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	-	
Operaciones	Usuario	Sistema
1		Lista autores registrados
2	Introduce datos de la obra	
3	Escoge autores de la obra	
4		Registra una obra con los datos proporcionados
5		Establece las autorías pertinentes

5.2.2.2.2. Modificar obra

Como en el caso de la modificación de autores, cambiaremos los datos referentes a una obra existente en el sistema. Además, podremos eliminar autores de la obra y/o añadirlos. El sistema guardará los cambios realizados y actualizará las autorías pertinentes.

Caso de uso	**Modificar obra**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	La obra existe	
Operaciones	Usuario	Sistema
1		Lista obras registradas
2	Escoge una para modificar	
3		Muestra los datos de la obra
4		Muestra autores disponibles para añadir a la obra
5	Modifica los datos oportunos	
6		Guarda los datos modificados para la obra
Extensiones		
6.A		Si el usuario ha añadido un autor a la obra, establece la autoría
6.B		Si el usuario ha eliminado un autor de la obra, borra la autoría

5.2.2.2.3. Borrar obra

El borrado de una obra implica que el sistema elimine la obra seleccionada, así como las autorías relacionadas con la obra.

Caso de uso	**Borrar obra**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	La obra existe	
Operaciones	Usuario	Sistema
1		Lista obras registradas
2	Escoge una para borrar	
3		Borra la obra seleccionada
4		Borra las autorías pertinentes

5.2.2.2.4. Consultar obra

A la hora de consultar una obra, el sistema nos proporcionará la información registrada para la obra escogida por el usuario, y los autores que figuran como tales para la misma.

Caso de uso	**Consultar obra**	
Actor primario	Usuario	
Sistema	Biblioteca	
Nivel	Objetivo usuario	
Condiciones previas	La obra existe	
Operaciones	Usuario	Sistema
1		Lista obras registradas
2	Escoge una para consultar	
3		Muestra los datos de la obra
4		Muestra las autorías de la obra

5.2.2.3. Mantenimiento de autorías

Los listados aquí son los casos de uso relativos al mantenimiento de autorías en el sistema. Se trata de casos de uso para los que el usuario no actúa como actor principal, ya que son casos de uso que son utilizados por el sistema en otros casos de uso.

5.2.2.3.1. Establecer autoría

Se trata de crear un nuevo vínculo entre un autor y una obra para identificar al primero como autor de la segunda.

Caso de uso	**Establecer autoría**
Sistema	Biblioteca
Nivel	Subfunción
Condiciones previas	Se proporciona un autor y una obra
Operaciones	Sistema
1	Relaciona el autor y la obra proporcionadas.

5.2.2.3.2. Eliminar autoría

En este caso, la finalidad es la inversa, desvincular a un autor de una obra.

Caso de uso	**Eliminar autoría**
Sistema	Biblioteca
Nivel	Subfunción
Condiciones previas	Se proporciona un autor y una obra
Operaciones	Sistema
1	Elimina la relación entre el autor y la obra proporcionadas.

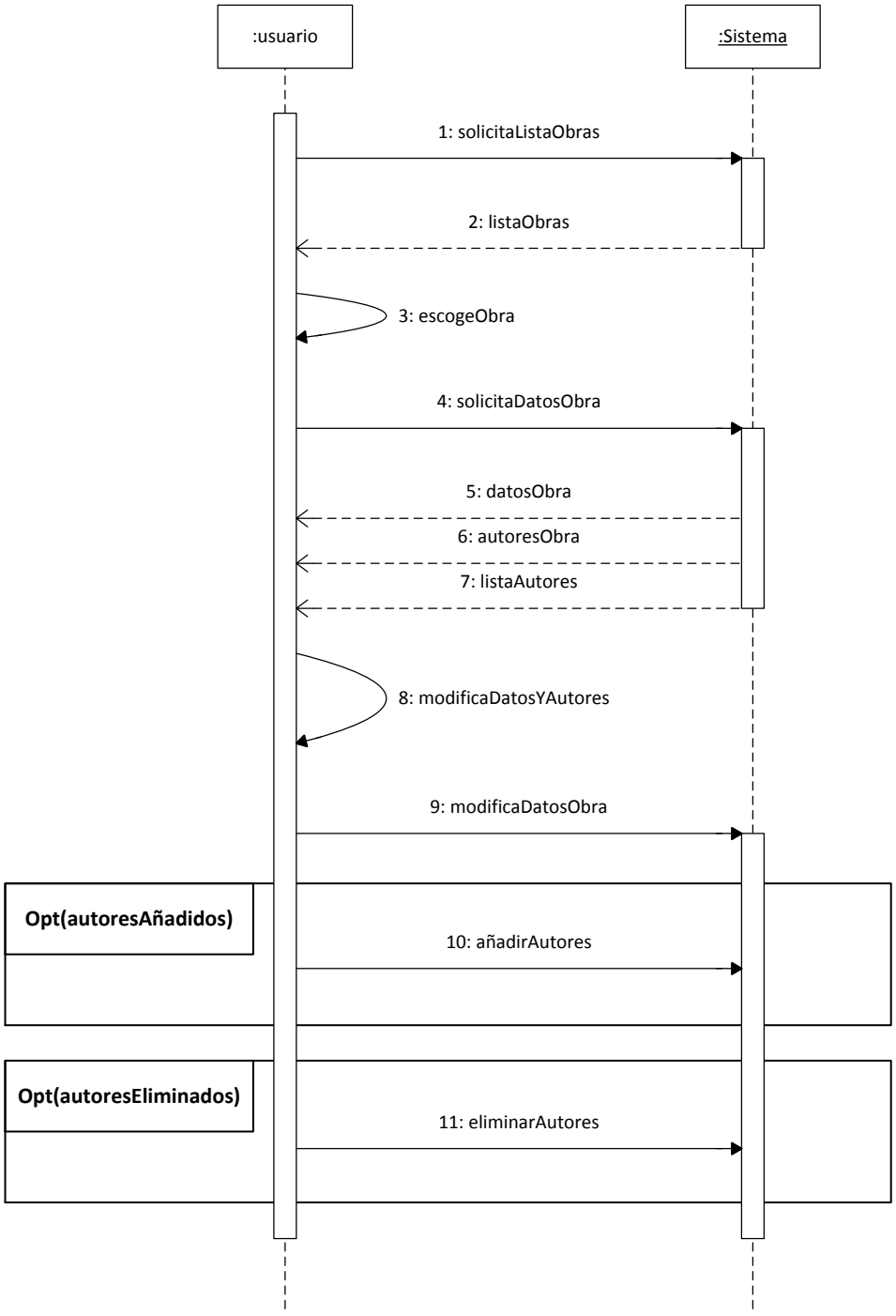
5.2.2.3.3. Consultar autoría

En el caso de la consulta, tendremos que indicar unos parámetros a partir de los cuales recuperar las autorías. Si queremos conocer las obras en las que participa un autor tendremos que indicar el autor en cuestión, y viceversa.

Caso de uso	**Consultar autoría**
Sistema	Biblioteca
Nivel	Subfunción
Condiciones previas	Se proporciona un autor o una obra
Operaciones	Sistema
1.A	Si se proporciona un autor
1.A.1	Se devuelven las obras relacionadas con el autor proporcionado
1.B	Si se proporciona una obra
1.B.1	Se devuelven los autores relacionados con la obra proporcionada

5.3. Diagramas de secuencia

A continuación exponemos el diagrama de secuencia correspondiente al caso de uso "Modificar obra". El resto de los casos de uso presentan un diagrama muy parecido (ya hemos visto la similitud entre los casos de uso en sus correspondientes representaciones textuales), por lo que no los detallaremos.


:usuario
:Sistema
1: solicitaListaObras
2: listaObras
3: escogeObra
4: solicitaDatosObra
5: datosObra
6: autoresObra
7: listaAutores
8: modificaDatosYAutores
9: modificaDatosObra
Opt(autoresAñadidos)
10: añadirAutores
Opt(autoresEliminados)
11: eliminarAutores

6. Diseño

Tal como adelantábamos en la solución propuesta, en su apartado dedicado al plano técnico, la implementación del sistema se llevará a cabo por módulos. Un primer módulo estará dedicado a la explotación de los datos. El segundo expondrá unos servicios web. El tercer y último módulo proveerá las interfaces gráficas con las que interactuará el usuario.

Adelantamos aquí, porque se verá en el apartado de implementación, que el sistema se implementará en Java.

Veamos cada uno de estos módulos en detalle.

6.1. Módulo de datos

En este módulo se realizarán todas aquellas operaciones destinadas a la persistencia de los datos. Es obvio, por tanto, que el elemento básico de este módulo lo ha de constituir un repositorio persistente de datos, una base de datos.

6.1.1. Base de datos

Utilizaremos como base de datos un SGBD[4] relacional. El uso de este tipo de base de datos en detrimento de otro tipo viene dado por la extensión de su uso, y sobre todo por la experiencia previa del alumno con ellas.

De cara a posibles mejoras del sistema, podría estudiarse el rendimiento del mismo con el uso de bases de datos NoSQL[5], aunque a priori no sería especialmente relevante por el hecho de que la aplicación no contará con un volumen especialmente grande de datos a almacenar.

6.1.2. Servicios

Por encima de esta base de datos, será necesario implementar unos servicios que provean al resto de la aplicación una vía para acceder y escribir en esta base de datos.

En primer lugar, realizaremos un mapeo entre los objetos que manejaremos en la aplicación y los datos relacionales de la base de datos. Esto lo conseguiremos utilizando un

[4] Sistema de Gestión de Bases de Datos

[5] Las bases de datos NoSQL difieren de las relacionales en el modo de almacenar la información. Mientras que las bases de datos relacionales requieren estructuras fijas en forma de tablas para el almacenamiento de datos, las NoSQL no. Otras diferencias son el lenguaje de consulta, las operaciones disponibles sobre los datos, etc. Sin embargo, las diferencias más importantes se encuentran en el rendimiento que ofrecen unas y otras en determinados escenarios, principalmente cuando la carga de datos que han de soportar aumenta.

sistema de ORM[6] que nos realice la conversión con los ajustes y programación que llevemos a cabo.

Hecho este mapeo, implementaremos los servicios antes mencionados, siguiendo el patrón DAO[7]. Este patrón define un método estándar de acceso y escritura a los datos, abstrayendo del modo en el que éstos se estén almacenando.

El patrón DAO nos proporcionará la flexibilidad necesaria para, en un futuro, cambiar el modo en el que persistir los datos si es necesario, sin afectar a la implementación de la aplicación.

6.2. Módulo de servicios web

Ya que nuestra aplicación proveerá al usuario de distintas interfaces para distintas plataformas (como veremos más adelante) para la consulta de los datos almacenados en el sistema, se perfila conveniente la implementación de un sistema que proporcione un acceso común a los servicios. Esto es, que cada una de las interfaces no conlleve replicar el acceso a los datos del sistema.

[6] Object-Relational Mapping
[7] Data Access Object, Objeto de Acceso a Datos

Este acceso común lo conseguiremos mediante la exposición de unos servicios web. Estos servicios web expondrán unas funciones de mantenimiento de las entidades presentes en el sistema, haciendo uso interno de los servicios expuestos en el punto anterior.

Los servicios web serán de tipo REST[8]. La elección de este tipo de servicios web sobre SOAP[9], que es otro tipo de servicio web muy extendido también, se basa en varios criterios que hemos tenido en cuenta para su elección.

El primero de ellos es la ligereza del tráfico que requiere REST. Toda la información requerida en la invocación de un servicio web está implícita en la URL accedida por la aplicación que está invocando el servicio, y el contenido que se está enviando. Este contenido no está fuertemente estructurado, sino que basta con utilizar XML[10] o JSON[11] sin la necesidad de definir esquemas. Además, la comunicación entre los dos puntos se realiza por HTTP, siendo este protocolo la base de REST.

El segundo aspecto, y quizás el más importante, es que SOAP conlleva una carga de procesado considerable. Quizás no sea considerable en el caso de una interfaz web que se está sirviendo desde un servidor, pero sí lo es si la llamada al servicio web se está realizando desde un Smartphone Android, donde los recursos son mucho más limitados.

SOAP requiere que el cuerpo del mensaje que se intercambia entre los dos puntos implicados en la llamada a un servicio web, el emisor (que es quien lo invoca) y el receptor (que es quien lo expone), siga unas determinadas directrices, haciendo que para la interpretación del mensaje sea necesario un procesado del mismo, así como la codificación y decodificación de las instancias de los tipos que hayamos incluido en el mismo.

REST es, por tanto, un protocolo más sencillo y liviano que se ajusta mucho más a las necesidades y características de nuestro sistema.

Para más información sobre los servicios web REST ver (Burke, 2010).

---

[8] *Representational State Transfer*, Transferencia de Estado Representacional
[9] *Simple Object Access Protocol*, Protocolo Simple de Acceso a Objetos
[10] *Extensible Markup Language*
[11] *JavaScript Object Notation*, un lenguaje de notación de objetos aún más simple que XML.

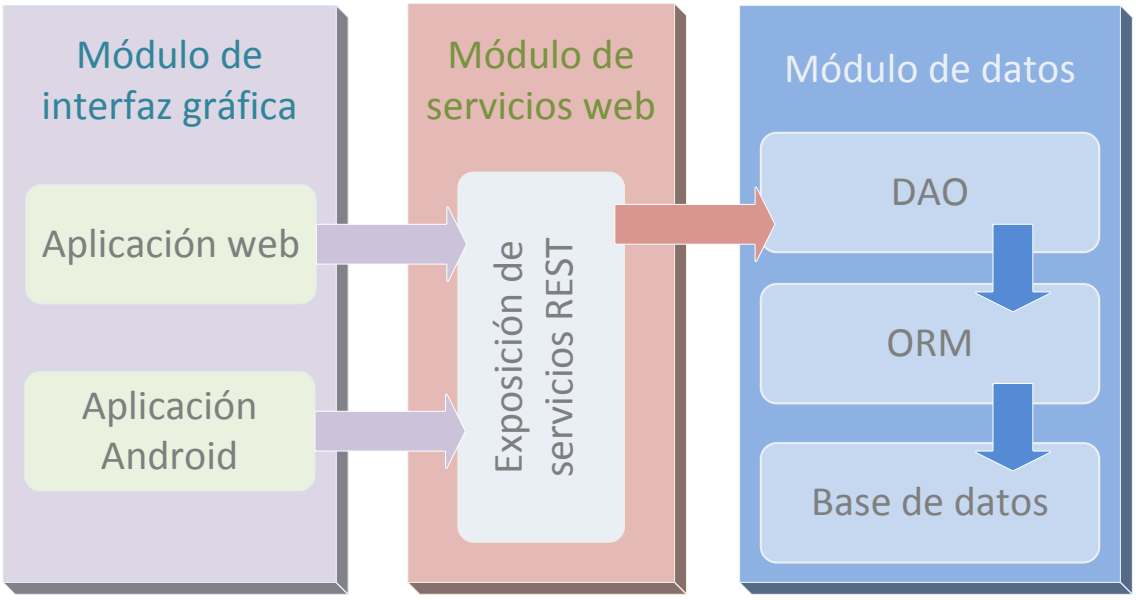
6.3. Módulo de interfaz gráfica

Como ya comentamos antes, se proveerán dos interfaces distintas. Una interfaz en forma de aplicación web, que proporcionará acceso a todas las funcionalidades del sistema; y otra interfaz servida por una aplicación Android, que permitirá las operaciones de consulta. Ambas aplicaciones accederán a los mismos servicios web que hemos expuesto antes.

La implementación de la aplicación web será, como antes hemos dicho, en Java. El método concreto de implementación lo veremos en el apartado correspondiente. En lo que a la aplicación Android respecta, ésta será una aplicación completamente nativa que correrá directamente sobre el teléfono en la que esté instalada. No se usarán por tanto frameworks similares a PhoneGap[12] que construyen aplicaciones híbridas, ni se implementará una aplicación web adaptada al tamaño de las pantallas de los dispositivos móviles.

[12] phonegap.com, es un framework para la creación de aplicaciones móviles híbridas. El desarrollador crea la aplicación haciendo uso de HTML, CSS y Javascript. El framework crea después distintas versiones para los distintos sistemas operativos móviles introduciendo el código necesario para renderizar y ejecutar el desarrollo realizado.

Vistos ya cada uno de los módulos, veamos esquemáticamente cómo quedaría configurado el sistema, desde un punto de vista arquitectónico, y los accesos que se producen entre los módulos.



Este diseño modular de la aplicación permite realizar cambios en la misma de una manera más sencilla, reduciendo el acoplamiento entre los distintos componentes del sistema. El

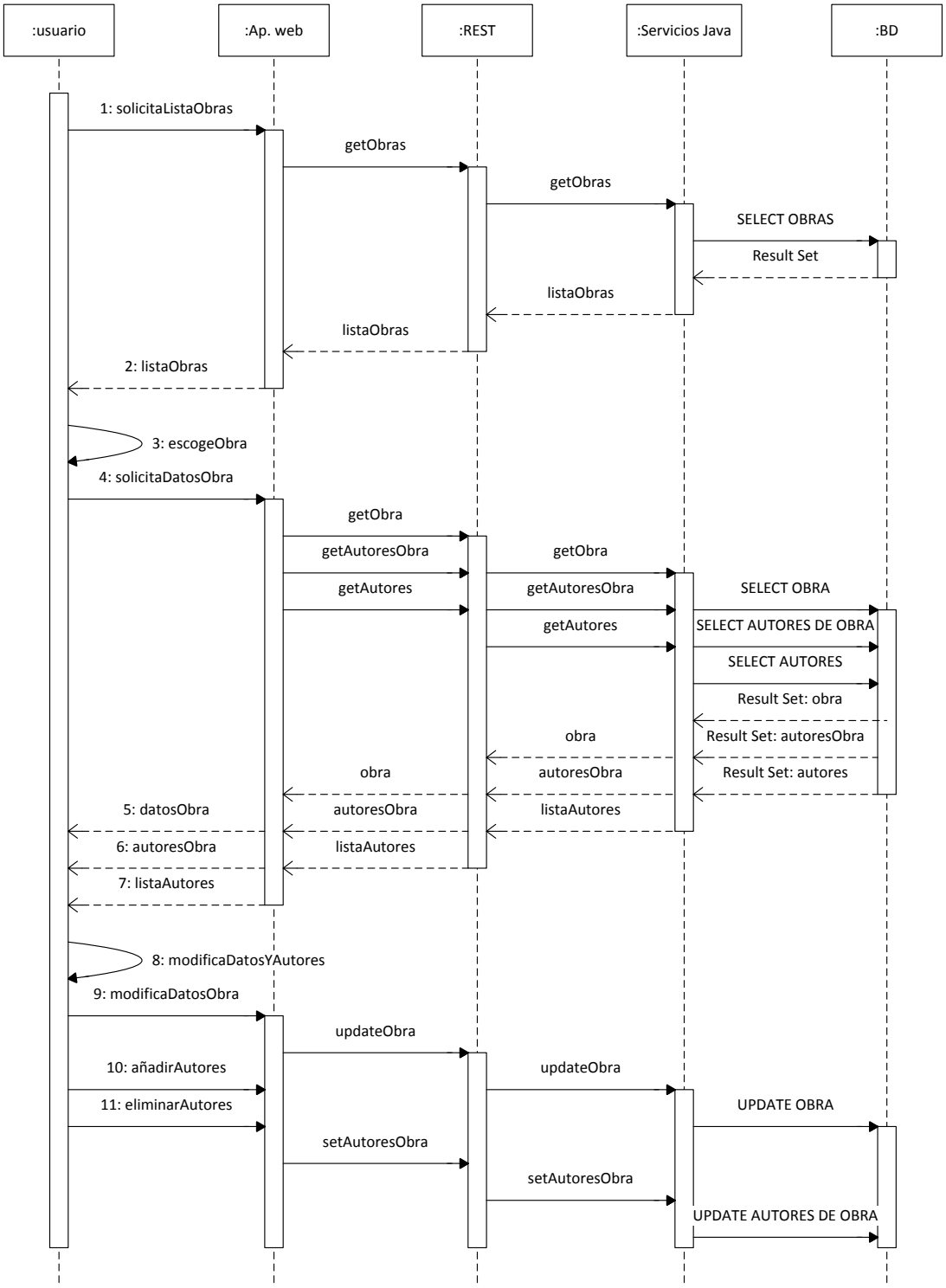
hecho de exponer unos servicios web que provean la lógica correspondiente a la explotación de los datos reduce aún más ese acoplamiento.

Un cambio en el sistema puede suponer modificar tan sólo uno de los módulos si los cambios así lo permiten. Puede bastar, por ejemplo, con añadir un nuevo servicio web con nueva lógica para explotar en otro orden los datos sin que se vean afectadas las capas de persistencia ni las capas de presentación (salvo para incorporar las nuevas funcionalidades de este servicio web).

En definitiva, ante un cambio a realizar, el sistema modular nos permite modificar tan sólo los módulos afectados por el cambio, no es necesaria una refactorización completa del sistema.

6.4. Diagrama de secuencia

Para ilustrar la dinámica de este diseño recuperaremos el diagrama de secuencia expuesto al final del análisis de requisitos. Lo extenderemos para ver de qué manera interactúan, a nivel general, los componentes presentes en los módulos.


:usuario
:Ap. web
:REST
:Servicios Java
:BD
1: solicitaListaObras
getObras
getObras
SELECT OBRAS
Result Set
listaObras
listaObras
2: listaObras
3: escogeObra
4: solicitaDatosObra
getObra
getAutoresObra
getAutores
getObra
getAutoresObra
getAutores
SELECT OBRA
SELECT AUTORES DE OBRA
SELECT AUTORES
Result Set: obra
Result Set: autoresObra
Result Set: autores
obra
autoresObra
listaAutores
obra
autoresObra
listaAutores
5: datosObra
6: autoresObra
7: listaAutores
8: modificaDatosYAutores
9: modificaDatosObra
updateObra
10: añadirAutores
11: eliminarAutores
updateObra
UPDATE OBRA
setAutoresObra
setAutoresObra
UPDATE AUTORES DE OBRA

7. Entorno de desarrollo

Antes de pasar a los temas relacionados con la implementación propiamente dicha del sistema, trataremos algunos aspectos relacionados con el entorno en el que se llevará a cabo el desarrollo.

Todo el desarrollo se llevará a cabo en el equipo del alumno. A modo de referencia, las especificaciones del equipo son las siguientes:

Intel Core i5-2410M a 2,3GHz, 64 bits
8 GB RAM
Windows 7 Professional 64 bits

7.1. JDK

Ya que el desarrollo se realizará en Java, es primordial tener instalada la última JDK[13] de la versión de Java para la cual vamos a desarrollar. En este caso, se utilizará la versión 7 de Java. En el momento de redactar esta memoria la última versión de la JDK para la versión 7 es la 7u25.

7.2. Entorno de desarrollo integrado

Como IDE[14] utilizaremos Eclipse[15]. La principal razón para utilizar Eclipse y no otro entorno de desarrollo es la gran familiaridad del alumno con este entorno.

De entre las varias versiones disponibles de Eclipse utilizaremos la denominada "Eclipse IDE for Java EE Developers". Esta versión incluye, además del entorno básico de desarrollo, una serie de plugins preinstalados útiles a la hora de desarrollar en el marco de trabajo que establece el estándar EE de Java.

7.3. Máquinas virtuales

Algunos de los elementos implicados en el desarrollo del sistema se instalarán en una máquina virtual en lugar de hacerlo en la propia máquina del alumno. Para disponer de esta máquina virtual se utilizará un software de virtualización.

El software escogido es VirtualBox[16] de Oracle. Otro software también muy conocido es VMware[17], pero se ha escogido VirtualBox por su sencillez de uso y porque la creación de máquinas virtuales es más rápida que en VMware. Se creará una máquina virtual con las siguientes características:

---

[13] *Java Development Kit*. Puede encontrarse en http://www.oracle.com/technetwork/java/javase/downloads/index.html
[14] *Integrated Development Environment*, Entorno de Desarrollo Integrado
[15] eclipse.org
[16] www.virtualbox.org
[17] www.vmware.com/es/

1 CPU con límite de ejecución del 100%
2 GB RAM
Adaptador a red de tipo puente (*bridged*)
Ubuntu 12.04.1 LTS 64 bits

7.4. Sistema de control de versiones

A modo de repositorio de código se utilizará Subversion (SVN)[18]. Existen otras alternativas como CVS[19] o GIT[20].

En el caso de CVS, se trata de un sistema anticuado, ya superado por SVN y GIT.

GIT es un sistema muy en auge, que ofrece una funcionalidad muy amplia consumiendo pocos recursos. Sin embargo, está enfocado principalmente al trabajo en equipo, aunque se puede usar también en equipos de una sola persona.

La elección de SVN sobre estos otros dos sistemas de control de versiones se debe, en primer lugar, a que habrá un solo desarrollador. Por tanto, los conflictos a la hora de realizar subidas de código serán inexistentes.

Por otra parte, la experiencia previa con SVN es muy amplia, mientras que no lo es con GIT.

Instalaremos SVN en la máquina virtual antes mencionada. Puede consultarse el procedimiento de instalación en el anexo C.

7.5. Gestión y construcción de proyectos Java

Por último, a la hora de construir los proyectos Java y de gestionar las dependencias, utilizaremos Maven[21].

En lo que se refiere a la construcción de proyectos existe como alternativa Ant[22]. Sin embargo, Maven ofrece una funcionalidad mayor, es más flexible y permite, sobre todo, realizar una gestión de dependencias muy eficaz, cosa que Ant no permite en absoluto.

Para más información sobre Maven ver el anexo B.

---

[18] http://subversion.apache.org/, se trata de uno de los proyectos de Apache.
[19] http://www.cvshome.org/, Concurrent Version System
[20] http://git-scm.com/, diseñado por Linus Torvalds.
[21] maven.apache.org, otro proyecto de Apache.
[22] ant.apache.org, también perteneciente a Apache.

8. Implementación del sistema

Abordamos en este apartado la implementación que hemos hecho del sistema de acuerdo con lo expuesto en el apartado correspondiente al alcance del proyecto. Seguiremos un orden secuencial, en consonancia con el orden seguido en el desarrollo. Este orden ha partido de lo más básico en el sistema (básico en el sentido del sustento de la aplicación), siguiendo el concepto modular expuesto al tratar de la arquitectura del sistema. Así pues, hemos comenzado por la base de datos para acabar en las interfaces.

Aunque hemos hablado de una secuencialidad en el desarrollo, según surgían problemas durante el desarrollo de la aplicación se han ido revisando los desarrollos anteriores para atajar estos problemas. Veremos explicados estos problemas en los apartados correspondientes a los puntos del desarrollo que presentaron los conflictos.

Un elemento muy importante del proyecto, sustancial, ha sido esa orquestación de tecnologías de la que hablábamos a la hora de explicar el objetivo de este PFC. A lo largo de este apartado no sólo se hablará de la implementación pura del sistema, sino que se tratará, antes del uso de cualquier producto o framework, de las decisiones tomadas para su elección, de cuáles eran sus alternativas y por qué fueron descartadas.

Un criterio que se extiende a lo largo de todo el proyecto es la primacía del Open Source sobre otras soluciones de tipo propietario.

8.1. Módulo de datos

Comenzaremos por el módulo de datos. Recordemos que, este módulo se encarga de todas aquellas operaciones relacionadas con la persistencia de datos en el sistema. El elemento básico de este módulo no es otro que la base de datos.

8.1.1. Base de datos

8.1.1.1. SGBD

Cuando tratábamos de la arquitectura del sistema, ya dijimos que el SGBD a utilizar sería de tipo relacional. En base a la experiencia del alumno con este tipo de bases de datos, los productos que se presentan como candidatos a ser implantados son dos: MySQL[23] y Oracle Database. Ambos son productos de Oracle, desde que en 2009 Oracle adquirió Sun Microsystems, quien a su vez había adquirido MySQL AB un año antes.

Las respectivas versiones que se perfilan como candidatas son:

- MySQL Community Server 5.5.31
- Oracle Database Express Edition 11g Release 2

[23] www.mysql.com

Ambas versiones son gratuitas, y son las últimas versiones disponibles en el momento de implantar la base de datos en el tiempo de desarrollo.

Oracle Database Express Edition es una versión reducida del producto *senior*, la edición Enterprise, y no requiere licencia de pago para su uso. Oracle Database es un software de licencia privativa. Oracle distribuye también, de manera gratuita, una herramienta a modo de GUI[24]. Se trata de SQLDeveloper[25], una herramienta sencilla que permite, entre otras muchas funcionalidades, la exploración gráfica de los datos almacenados en bases de datos locales o remotas, exportación e importación de datos y gestión de usuarios.

MySQL Community Server es una base de datos con licencia GNU GPL[26]. Se trata de un producto más sencillo y liviano en cuanto a necesidades de recursos hardware que Oracle Database.

Como GUI, MySQL dispone de MySQL Workbench[27], un producto muy completo para lo que son las posibilidades de MySQL. Permite la gestión de bases de datos locales y remotas, la exploración gráfica de las mismas, y una completa *suite* de diseño y modelado de bases de datos.

Existen entre ambos productos diferencias funcionales que se hacen evidentes a la hora de trabajar con ellos. El dialecto SQL que utilizan no es el mismo, por ejemplo. Otra diferencia, que juega muy a favor de Oracle Database, es la presencia en éste de las vistas materializadas, función que en MySQL se ha de simular con el uso de tablas auxiliares y triggers programados sobre las tablas referenciadas por la vista.

Sin embargo, estas diferencias no son relevantes para el caso que nos ocupa. En el caso de las dos comentadas, el dialecto SQL no nos afecta en absoluto ya que, como veremos más adelante, un ORM se encargará de la comunicación con la base de datos. Respecto a las vistas materializadas, nuestra base de datos no presentará ni una estructura ni un volumen de datos que justifique su uso.

Tomando todas estas consideraciones, implantaremos MySQL como base de datos del sistema, ya que se ajusta más a las características de nuestro sistema. Resumiendo, las razones son:

- Licencia abierta
- Mayor facilidad de uso
- Menor requerimiento de recursos hardware

---

[24] *Graphic User Interface*, Interfaz Gráfica de Usuario
[25] http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html
[26] GNU General Public License, declara un software como libre y cualquier producto que derive del mismo ha de distribuirse bajo la misma licencia.
[27] http://dev.mysql.com/downloads/tools/workbench/5.2.html

- MySQL Workbench nos proporciona una utilidad muy valiosa que utilizaremos como veremos a continuación.

Comentaremos por último que la base de datos se instalará en la máquina virtual que utilizaremos a lo largo del desarrollo. Esta instalación puede verse en el anexo C.

8.1.1.2. Modelado de la base de datos

Para el modelado de datos utilizaremos la herramienta MySQL Workbench que hemos mencionado antes, que nos permitirá hacer un diagrama EER[28] de manera intuitiva para posteriormente, con la funcionalidad incluida de *forward engineering*, obtener un script SQL de creación del esquema de base de datos y ejecutarlo en una instancia de MySQL determinada.

A la hora de diseñar este diagrama EER, nos retrotraeremos al diagrama de clases expuesto en el análisis de requisitos. Básicamente, trasladaremos este diagrama de clases a un diagrama EER.

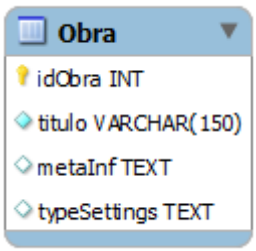
Primero abordaremos las entidades, con la conversión de los tipos de los atributos, para más tarde abordar la conversión de las relaciones entre entidades y las tablas y atributos que se crearán en función de las multiplicidades de estas relaciones.

8.1.1.2.1. Atributos

Antes de tratar cada una de las entidades, huelga decir que todas las entidades contarán con un identificador de tipo entero, que será asignado por la propia base de datos en el momento de la inserción de registros. Este identificador actuará como clave primaria única en la mayoría de los casos.

8.1.1.2.1.1. Obra

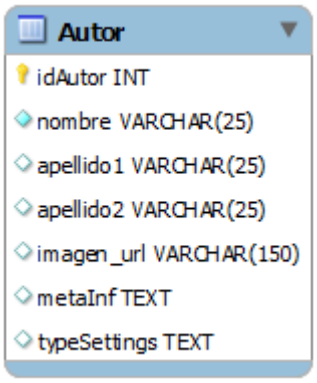
Los tres atributos que presenta esta entidad son de tipo cadena de caracteres (en adelante string). Aquí los sustituiremos por dos tipos de datos de MySQL distintos. TEXT para metaInf y typeSettings (estos tipos serán también utilizados para estos atributos en el resto de entidades que los presentan), mientras que para el título definiremos un tipo VARCHAR de 150 posiciones.



[28] *Enhanced Entity-Relationship model*, o *Extended Entity-Relationship model*, modelo de Entidad-Relación Extendido

8.1.1.2.1.2. Autor

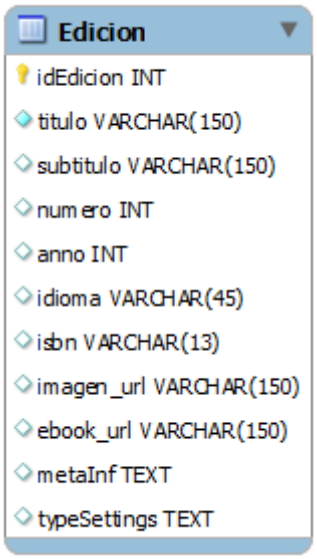
Para el autor, los atributos de tipo string se sustituyen por VARCHAR. En el caso del nombre y los dos apellidos, de 25 posiciones. Para la url de la imagen, 150.



8.1.1.2.1.3. Edición

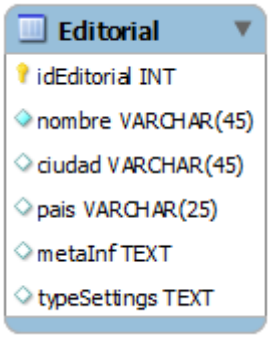
Esta entidad es la que más atributos presenta. Los atributos de número y año toman el tipo Integer de MySQL. Nótese que el nombre del atributo correspondiente al año pasa a llamarse anno, debido a que la ñ es un carácter especial.

El resto de atributos pasan a ser de tipo VARCHAR, aunque de distintas longitudes. El título, subtítulo y las dos URL's acepatarán hasta 150 caracteres. El idioma, 45. EL ISBN acepta hasta 13, que es su longitud máxima en la actualidad.



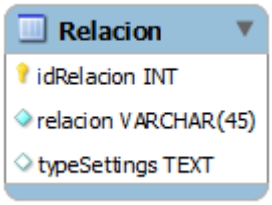
8.1.1.2.1.4. Editorial

Todos los atributos de la editorial se convierten de string a VARCHAR. Las longitud de este VARCHAR será de 45 caracteres, salvo para el país, que será de 25.



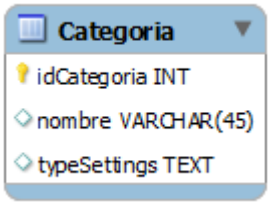
8.1.1.2.1.5. Relación

El atributo que define la relación, que es de tipo string, pasa a ser VARCHAR de 45 posiciones. Prestaremos más atención a esta entidad cuando tratemos la relación entre las entidades Autor y Edición.



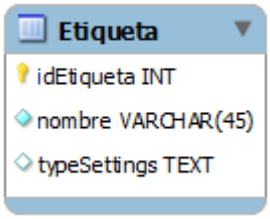
8.1.1.2.1.6. Categoría

La Categoría sólo presenta un atributo: nombre. Este atributo pasa a ser un VARCHAR, también de 45 posiciones.



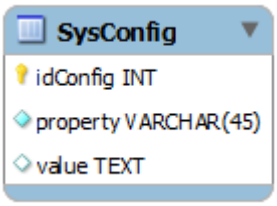
8.1.1.2.1.7. Etiqueta

Al igual que la Categoría, tan sólo se almacena su nombre, que recibe como tipo de atributo la misma conversión.



8.1.1.2.1.8. SysConfig

Esta entidad no estaba presente en nuestro diagrama de clases. La única finalidad de la tabla SysConfig es la de almacenar propiedades de configuración del sistema en forma de clave-valor. En la implementación que haremos en este proyecto no se utiliza ninguna propiedad, pero en el futuro es muy probable que surgiera la necesidad de utilizar alguna. Imaginemos, por ejemplo, que deseamos implementar un backup periódico de los datos almacenados que los exporte a una hoja de cálculo. Esta tabla nos será útil para definir parámetros como la periodicidad de ese backup o el formato de salida para esa hoja de cálculo.

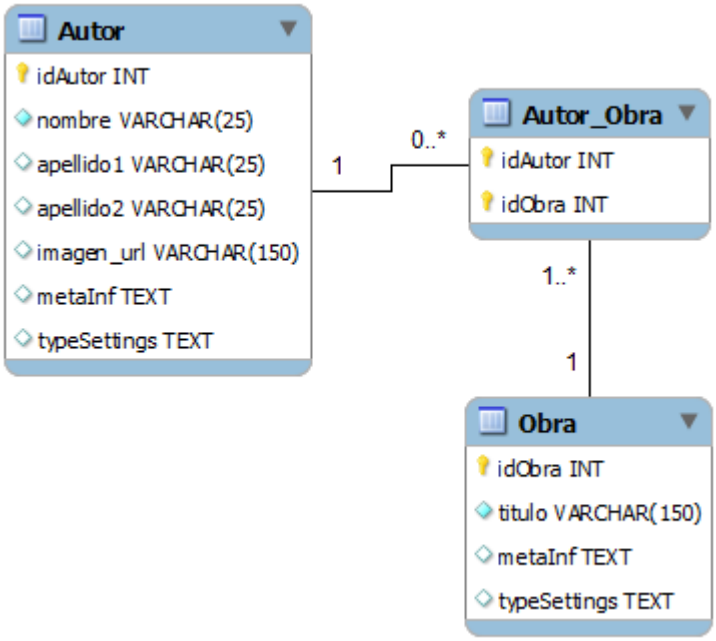


8.1.1.2.2. Relaciones

Adaptaremos ahora las relaciones definidas en nuestro diagrama de clases. Lo haremos en el orden en el que las tratábamos en el análisis de requisitos.

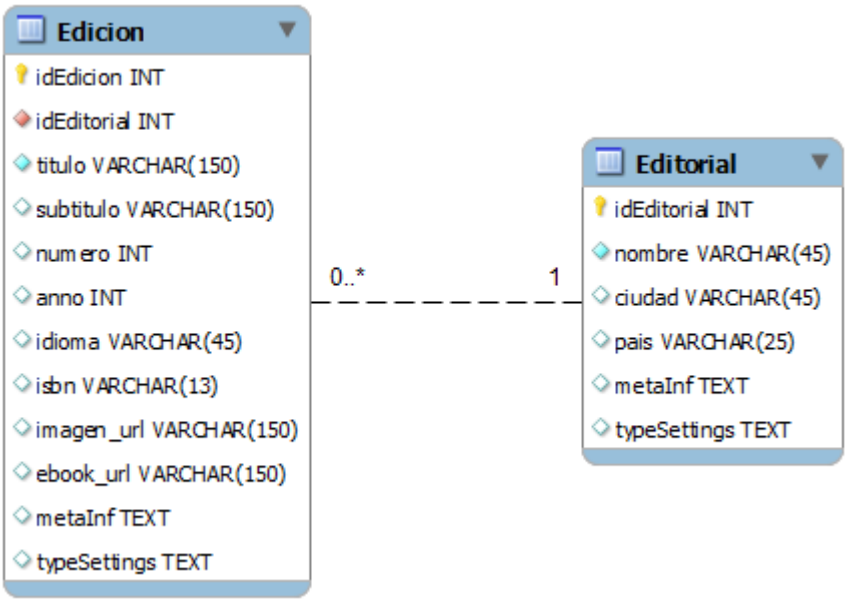
8.1.1.2.2.1. Autor-Obra

La cardinalidad 1..n-n de esta relación hace necesario utilizar una tabla que relacione ambas entidades. Denominaremos a esta tabla Autor_Obra, y contendrá dos únicas columnas con los respectivos identificadores del autor y la obra relacionadas. La conjunción de ambos identificadores actuará como clave compuesta primaria.



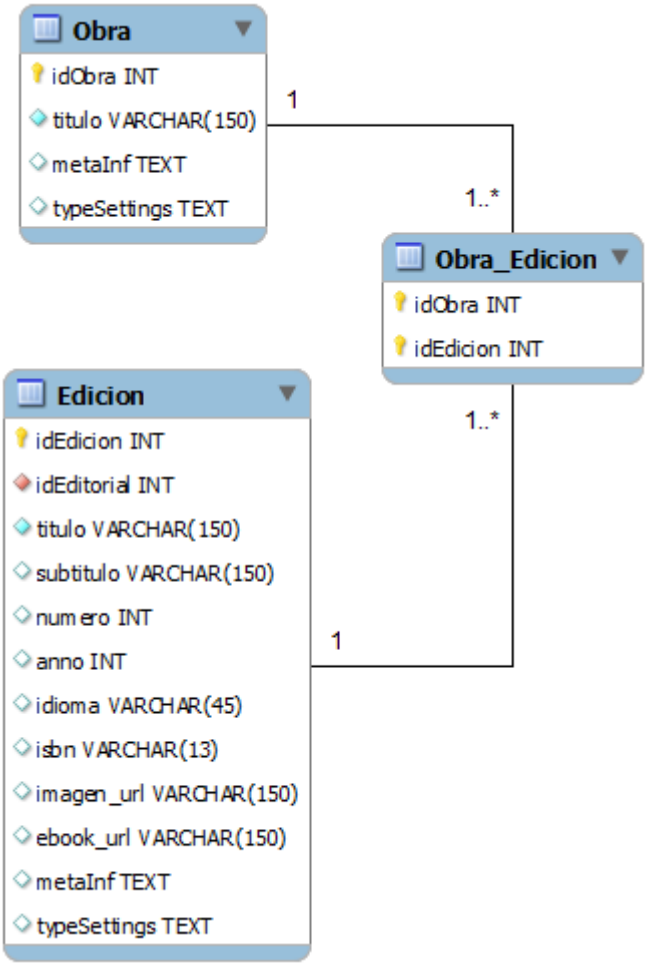
8.1.1.2.2.2. Edición-Editorial

Esta relación presenta una cardinalidad n-1, por lo que nos bastará incluir en la tabla Edición el identificador de la editorial como *foreign key*.



8.1.1.2.2.3. Obra-Edición

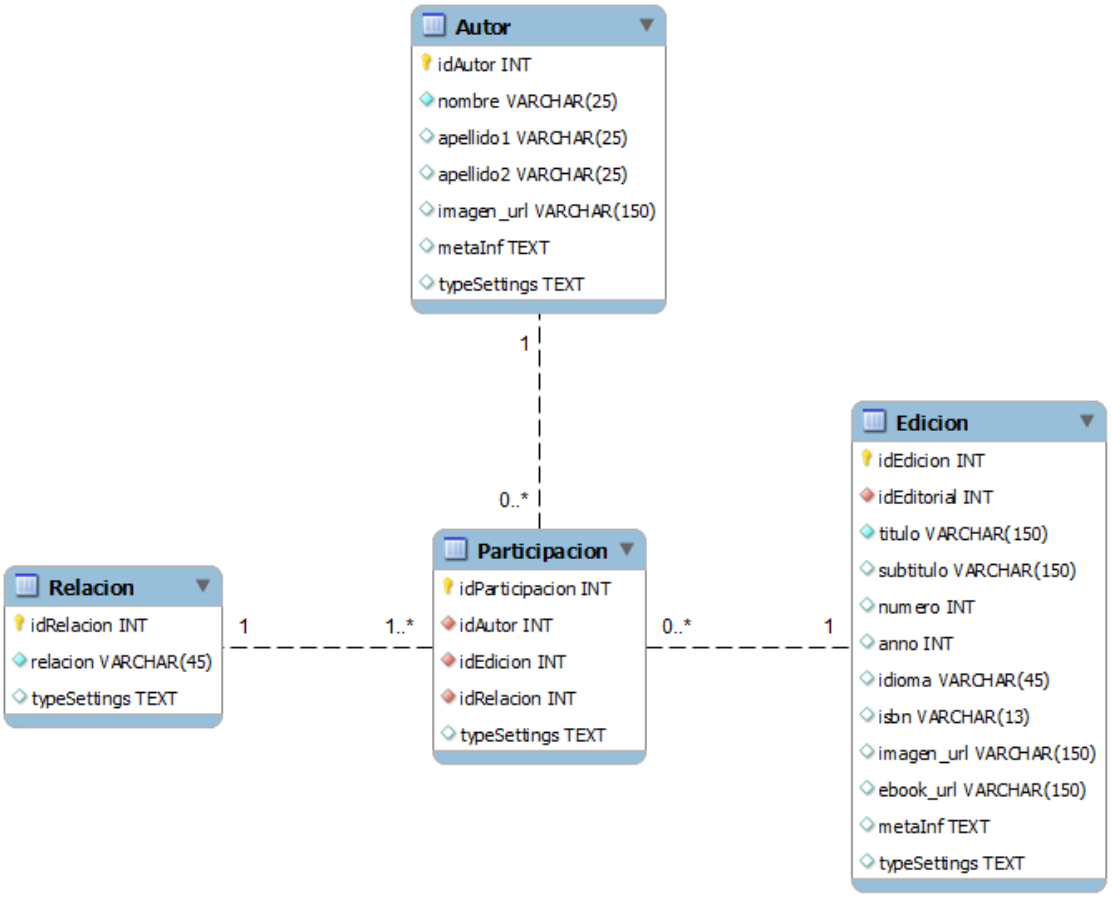
En este caso encontramos una relación 1..n-1..n, lo cual nos obliga de nuevo a crear una nueva tabla (Obra_Edicion) con los identificadores de ambas entidades actuando como clave compuesta primaria.



8.1.1.2.2.4. Autor-Edición-Relación(Participación)

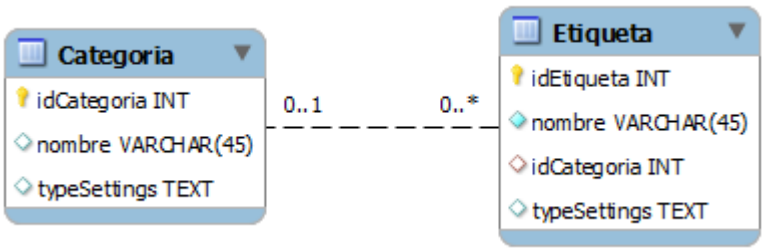
En el diagrama de clases, Relación figura como una clase asociación. Hemos creído conveniente transformarla, aquí en una tabla independiente que almacene los distintos tipos de relaciones presentes en el sistema. Se agilizarán así las búsquedas por tipo de relación y se establecerá un nombre común para las distintas formas de relación.

Esta relación pasa a ser, por tanto, ternaria. Se creará una tabla (Participación) conteniendo los identificadores de las entidades involucradas. Sin embargo, la tabla contendrá un identificador único a modo de clave primaria, no siendo ésta la conjunción de las otras tres claves. Esto es debido a que se simplifica el uso de la tabla por parte del ORM y su manejo en el código de la aplicación (concretamente, en el marshalling del objeto Participación).



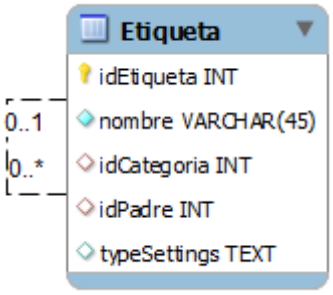
8.1.1.2.2.5. Categoría-Etiqueta

La cardinalidad de esta relación, 0..1-n, permite que baste con incluir una columna adicional a la tabla Etiqueta para almacenar el identificador de la categoría, si es que la etiqueta la tiene como etiqueta raíz.



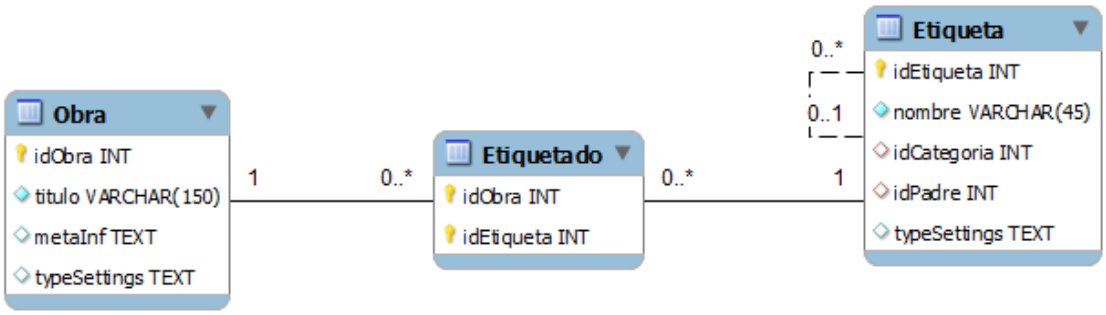
8.1.1.2.2.6. Etiqueta-Etiqueta

La jerarquización de etiquetas hace que esta entidad esté relacionada consigo misma. La cardinalidad coincide con la vista en la relación entre Categoría y Etiqueta, con lo que la solución es la misma que en el anterior caso, una nueva columna con el identificador de la etiqueta padre, si es que tiene padre.

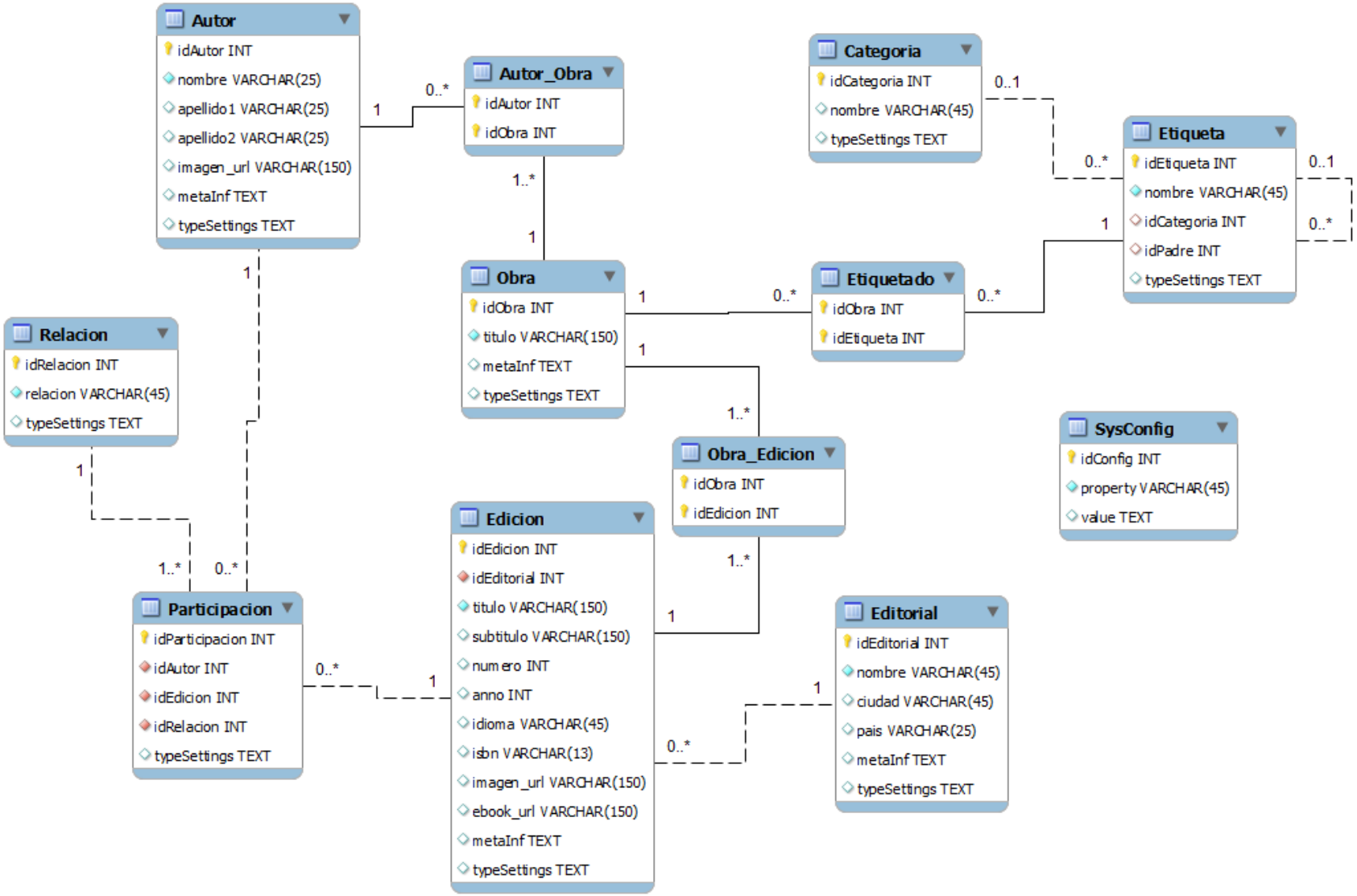


8.1.1.2.2.7. Obra-Etiqueta

La última relación es la que permite el etiquetado de obras. La cardinalidad aquí es n-n, con lo que es necesario crear una tabla con los identificadores de la obra y la etiqueta asignada. La tabla tendrá por nombre Etiquetado.



Con esta relación de etiquetado, termina la conversión del diagrama de clases a EER. El diagrama EER completo es el siguiente:


Autor
idAutor INT
nombre VARCHAR(25)
apellido1 VARCHAR(25)
apellido2 VARCHAR(25)
imagen_url VARCHAR(150)
metaInf TEXT
typeSettings TEXT
Autor_Obra
idAutor INT
idObra INT
Categoria
idCategoria INT
nombre VARCHAR(45)
typeSettings TEXT
Etiqueta
idEtiqueta INT
nombre VARCHAR(45)
idCategoria INT
idPadre INT
typeSettings TEXT
Obra
idObra INT
titulo VARCHAR(150)
metaInf TEXT
typeSettings TEXT
Etiquetado
idObra INT
idEtiqueta INT
Relacion
idRelacion INT
relacion VARCHAR(45)
typeSettings TEXT
Obra_Edicion
idObra INT
idEdicion INT
SysConfig
idConfig INT
property VARCHAR(45)
value TEXT
Participacion
idParticipacion INT
idAutor INT
idEdicion INT
idRelacion INT
typeSettings TEXT
Edicion
idEdicion INT
idEditorial INT
titulo VARCHAR(150)
subtitulo VARCHAR(150)
numero INT
anno INT
idioma VARCHAR(45)
isbn VARCHAR(13)
imagen_url VARCHAR(150)
ebook_url VARCHAR(150)
metaInf TEXT
typeSettings TEXT
Editorial
idEditorial INT
nombre VARCHAR(45)
ciudad VARCHAR(45)
pais VARCHAR(25)
metaInf TEXT
typeSettings TEXT

8.1.1.3. Generación de la base de datos

Una vez diseñado el diagrama, bastará con utilizar la herramienta de *forward engineering* de MySQL Workbench. Esta herramienta creará un script de creación para las tablas que ejecutaremos sobre la base de datos. El script generado, donde podemos ver también la creación del esquema, la codificación, comentarios y algunas restricciones que hemos omitido, se puede encontrar en el anexo D.

8.1.2. Modelo de datos

Establecido ya un esquema de base de datos, el siguiente paso fundamental es la traslación de las entidades en base de datos a nuestra aplicación. Para esto necesitamos establecer un mapeo entre los objetos que manejaremos en la aplicación y los registros presentes en la base de datos. Lo haremos utilizando un sistema de ORM.

8.1.2.1. ORM

Es evidente que no construiremos un sistema ORM existiendo ya herramientas tan extendidas. Ejemplos de ello son Hibernate[29] o EclipseLink[30]. La primera es especialmente conocida.

De hecho, Hibernate se perfila como la principal candidata a ser utilizada en nuestro sistema. Hibernate permite, mediante ficheros de configuración en XML o el uso de anotaciones Java directamente en las clases del modelo, definir la correspondencia entre el modelo usado en Java y el existente en base de datos.

Esencialmente, indicaremos para cada clase del modelo qué tabla de base de datos le corresponde, y con qué columnas se identifican los atributos de la clase. La idea general del mapeo es ésta, aunque las posibilidades son muchas más.

Hibernate se distribuye, nuevamente, con licencia GNU GPL, con lo que casa perfectamente con nuestra filosofía de dar prioridad a herramientas open source. Hibernate constituye además una implementación de JPA[31], la API de persistencia desarrollada para la plataforma Java EE y que proviene de EJB3[32]. Y ya que esto es así, decidimos establecer el mapeo ORM haciendo uso de la API definida en Java EE.

En este punto, establecemos otro criterio que podremos expandir a las decisiones a tomar respecto a las soluciones candidatas a ser adoptadas en lo que resta del módulo de datos y en el de exposición de servicios web. No es otro que primar el uso de las APIs existentes en la especificación EE de Java, más concretamente en la versión 6.

Así pues, usaremos JPA para establecer el mapeo ORM en el módulo de datos.

---

[29] www.hibernate.org, es parte de la familia de productos de JBoss
[30] www.eclipse.org/eclipselink
[31] En su versión 2.0 constituye la JSR 317, http://www.jcp.org/en/jsr/detail?id=317
[32] La versión 3 de Enterprise JavaBeans, otra API perteneciente a Java EE. Se trata de la JSR 220, http://www.jcp.org/en/jsr/detail?id=220

8.1.2.2. Generación del modelo

Escogido el modo en el que realizar el mapeo, el siguiente paso es crear las clases del modelo. Para ello utilizaremos una herramienta que nos automatice la generación las clases.

La herramienta en cuestión es Hibernate Tools[33]. Utilizaremos el plugin que se distribuye para Maven.

La característica principal de Hibernate Tools es la de realizar un *reverse engineering* para generar de manera automática las clases del modelo de dominio en forma de POJOs[34] desde una base de datos existente. El mapeo puede configurarse para que se realice por xmls de configuración o bien anotaciones sobre las propias clases. Las anotaciones pueden configurarse para que sean las propias de Hibernate, o las definidas por JPA (aunque Hibernate Tools las identifica con EJB3, lo que viene a ser lo mismo).

En nuestro caso, deseamos que el mapeo se realice a través de las anotaciones presentes en la propia clase.

Sin embargo, los POJOs generados de esta forma no concuerdan del todo con las clases que deseamos obtener. Esto es debido a que, adelantándonos a los acontecimientos, deseamos introducir un método de *marshalling*[35] que nos facilite la recepción y el envío de mensajes en los servicios web. En este apartado no veremos nada más referente a este tema que la configuración de las clases para facilitar ese proceso de *marshalling*.

8.1.2.2.1. JAXB

Este *marshalling* la realizaremos adoptando la API JAXB[36] de Java. JAXB permite, mediante la anotación de los atributos de una clase, establecer las directrices para su serialización en XML.

Con todo esto, lo que haremos será modificar en lo necesario la forma en la que Hibernate Tools genera las clases. Esta forma consiste en conectarse a una base de datos, y para cada una de las entidades que encuentra en el esquema al que se conecta, ejecutar unas plantillas, escritas en Freemarker[37], que generarán finalmente la clase del modelo que le corresponda. Lo que debemos hacer, por tanto, es modificar esas plantillas.

Las plantillas las hemos obtenido decompilando el JAR de la distribución de Hibernate Tools. En las plantillas hemos incluido la ruta de los paquetes de las anotaciones de JAXB

---

[33] http://www.hibernate.org/subprojects/tools.html, también de JBoss
[34] Plain Old Java Object, son clases consistentes en atributos de visibilidad privada y *getters* y *setters* para su acceso.
[35] El *marshalling* consiste en la serialización de objetos.
[36] Java Architecture for XML Binding. JAXB 2.0 es la JSR 222, http://www.jcp.org/en/jsr/detail?id=222
[37] http://freemarker.sourceforge.net/

que incluiremos y las directivas necesarias para que anote los atributos de la clase. Veamos un ejemplo de plantilla modificada:

```
<#-- // Fields -->

<#foreach field in pojo.getAllPropertiesIterator()><#if
pojo.getMetaAttribAsBool(field, "gen-property", true)> <#if
pojo.hasMetaAttribute(field, "field-description")>    /**
     ${pojo.getFieldJavaDoc(field, 0)}
     */
 </#if>   <#if
field.equals(clazz.identifierProperty)>@XmlElement<#else>@XmlElemen
t</#if> ${pojo.getFieldModifiers(field)}
${pojo.getJavaTypeName(field, jdk5)} ${field.name}<#if
pojo.hasFieldInitializor(field, jdk5)> =
${pojo.getFieldInitialization(field, jdk5)}</#if>;
</#if>
</#foreach>
```

Esta es la plantilla que se encarga de escribir los atributos de la clase. Como vemos, hemos incluido la anotación @XmlElement, que especifica que el atributo se ha de serializar de forma <nombreAtributo>valorAtributo</nombreAtributo>. Aunque en primer lugar tenemos que indicar que la clase es serializable en XML. Eso lo conseguimos anotando la clase, y para ello otra de las plantillas modificadas es la de la declaración de la clase:

```
/**
${pojo.getClassJavaDoc(pojo.getDeclarationName() + " generated by
hbm2java", 0)}
 */
<#include "Ejb3TypeDeclaration.ftl"/>
@XmlRootElement(name="${pojo.getDeclarationName()?uncap_first}")
@XmlAccessorType(XmlAccessType.FIELD)
${pojo.getClassModifiers()} ${pojo.getDeclarationType()}
${pojo.getDeclarationName()} ${pojo.getExtendsDeclaration()}
${pojo.getImplementsDeclaration()}
```

Aquí indicamos, con @XmlRootElement, que la clase es serializable en XML, y que la etiqueta raíz es el nombre de la clase en minúsculas (uncap_first). La anotación @XmlAccessorType sirve para indicar qué tipo de acceso a las propiedades queremos para obtener su valor. Aquí indicamos que consultaremos los campos de la clase directamente, no los getters, por ejemplo.

8.1.2.2.2. Hibernate Tools

Hechas estas modificaciones, configuramos Hibernate Tools para que se conecte a nuestra base de datos. Basta con crear un fichero de propiedades como el siguiente:

```
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.password=biblioteca
hibernate.connection.username=biblioteca
hibernate.default_schema=biblioteca
hibernate.connection.url=jdbc:mysql://vmPfc:3306/biblioteca
hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

Y configuramos la ejecución de la ingeniería inversa con el siguiente fichero xml, en el que indicaremos explícitamente las tablas a consultar e indicamos algunos nombre sustitutorios para algunas de las propiedades ya que, por defecto, la herramienta utilizará como nombres de atributos los que encuentre en la base de datos. El fichero es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate
Reverse Engineering DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-reverse-engineering-3.0.dtd" >

<hibernate-reverse-engineering>

	<table-filter match-catalog="biblioteca" match-name="Autor" />
	<table-filter match-catalog="biblioteca" match-name="Autor_Obra" />
	<table-filter match-catalog="biblioteca" match-name="Categoria" />
	<table-filter match-catalog="biblioteca" match-name="Edicion" />
	<table-filter match-catalog="biblioteca" match-name="Editorial" />
	<table-filter match-catalog="biblioteca" match-name="Etiqueta" />
	<table-filter match-catalog="biblioteca" match-name="Etiquetado" />
	<table-filter match-catalog="biblioteca" match-name="Obra" />
	<table-filter match-catalog="biblioteca" match-name="Obra_Edicion"
	/>
	<table-filter match-catalog="biblioteca" match-name="Participacion"
	/>
	<table-filter match-catalog="biblioteca" match-name="Relacion" />
	<table-filter match-catalog="biblioteca" match-name="SysConfig" />

	<table name="Autor">
		<foreign-key constraint-name="fk_Participacion.Autor">
			<set property="participaciones" />
		</foreign-key>
	</table>

	<table name="Edicion">
		<foreign-key constraint-name="fk_Participacion.Edicion">
			<set property="participaciones" />
		</foreign-key>
	</table>

	<table name="Editorial">
		<foreign-key constraint-name="fk_Edicion.Editorial">
			<set property="ediciones" />
		</foreign-key>
	</table>

	<table name="Etiqueta">
		<foreign-key constraint-name="fk_Etiqueta.Etiqueta">
			<many-to-one property="padre" />
			<set property="hijas" />
		</foreign-key>
	</table>

	<table name="Obra">
		<foreign-key constraint-name="fk_Obra_Edicion.Obra">
```

```
                <set property="ediciones" />
            </foreign-key>
            <foreign-key constraint-name="fk_Autor_Obra.Obra">
                <set property="autores" />
            </foreign-key>
        </table>

        <table name="Relacion">
            <foreign-key constraint-name="fk_Participacion.Relacion">
                <set property="participaciones" />
            </foreign-key>
        </table>

</hibernate-reverse-engineering>
```

Con estos dos ficheros, configuramos a través del plugin de Maven Hibernate Tools. Le indicamos los ficheros de configuración que debe leer, la ruta en la que dejar las clases generadas, el paquete al que deben pertenecer, la plantilla que debe ejecutar (en este caso es una plantilla que importa todas las demás en un orden secuencial), el patrón para el nombre de las clases y que genere código compilable con la versión 5 de Java y utilice anotaciones JPA. He aquí el extracto del pom.xml (el fichero que utiliza Maven para su ejecución):

```
<execution>
        <phase>generate-resources</phase>
        <id>PojoGenerator</id>
        <goals>
            <goal>hbmtemplate</goal>
        </goals>
        <configuration>
            <components>
                <component>
                    <name>hbmtemplate</name>
                    <outputDirectory>
                        src/main/java/es/lta/biblioteca/model
                    </outputDirectory>
                </component>
            </components>
            <componentProperties>
                <packagename>es.lta.biblioteca.model</packagename>
                <revengfile>/src/hibernate/hibernate.reveng.xml</revengfile>
                <propertyfile>
                    /src/hibernate/hibernate.properties
                </propertyfile>
                <configurationfile>
                    /src/hibernate/hibernate.cfg.xml
                </configurationfile>
                <implementation>jdbcconfiguration</implementation>
                <templatepath>src/hibernate/templates/pojo</templatepath>
                <template>Pojo.ftl</template>
                filepattern>{class-name}.java</filepattern>
                <ejb3>true</ejb3>
                <jdk5>true</jdk5>
            componentProperties>
        </configuration>
</execution>
```

Al ejecutar la tarea de Maven, se crearán todas las clases con las anotaciones JPA y JAXB. Sin embargo, hay algunos aspectos que no hemos podido controlar con las plantillas y que tendremos que modificar a mano una vez generadas de forma automática.

La primera modificación es referente a las relaciones entre entidades. Y abarca dos aspectos: uno, que define el comportamiento a seguir con las instancias de una clase referenciadas en caso de modificación de una instancia de la clase anotada, y que se indica con el atributo *cascade* de la anotación para la relación; y dos, que define cuándo se han de cargar las instancias referenciadas.

En el caso del *cascade*, Hibernate Tools genera el código de modo que se establece el valor CascadeType.ALL, es decir, que las instancias referenciadas se actualizarán del mismo modo que la instancia de la clase anotada. Entendámoslo mejor con un ejemplo, que nos llevará a ver la razón por la que queremos cambiarlo. Un autor tiene en su atributo 'obras' referenciadas todas las obras de las que es autor o co-autor. Si borramos este autor, se ejecutarán las operaciones en cascada que están anotadas en la relación. Ya que son todas (en JPA todas son: *persist, refresh, merge, detach* y *remove*), se borrarán también las obras referidas.

Por tanto, cambiaremos en las anotaciones de las relaciones CascadeType.ALL por {CascadeType.PERSIST,CascadeType.REFRESH,CascadeType.MERGE}.

En cuanto a la carga de instancias referenciadas, el *fetching*, Hibernate Tools lo establece a FetchType.LAZY, es decir, la instancia se añade al Entity Manager de JPA cuando la instancia es requerida para su lectura. Desgraciadamente, este comportamiento no es el esperado en la mayoría de los casos con JPA, y nos hemos visto obligados a establecerlo a FetchType.EAGER, que cargará todas las instancias referenciadas en el momento de cargar la instancia que las referencia.

La segunda modificación atañe a JAXB. A la hora de serializar un objeto en XML, se consultan todos sus atributos marcados como serializables. Si se trata de atributos de tipo básico (como int, long, etc. o sus *wraps* Integer, Long, etc.), Strings, o similares, se serializa su valor. Si, por el contrario, son clases que están a su vez anotadas con JAXB, se serializarán para obtener el valor del atributo. El problema se da cuando ambas clases se referencian mutuamente.

Por ejemplo. Imaginemos que queremos serializar la instancia de Autor que representa a Joseph Pearce. Para él sólo tenemos disponibles el nombre, un apellido y sus obras referenciadas. El XML resultante sería:

```
<autor>
        <idAutor>5</idAutor>
        <nombre>Joseph</nombre>
        <apellido1>Pearce</apellido1>
        <obras>
                <obra>
```

```
<idObra>17</idObra>
<titulo>Through Shakespeare's eyes</titulo>
<autors>
        <autor>
                <idAutor>5</idAutor>
                <nombre>Joseph</nombre>
                <apellido1>Pearce</apellido1>
                <obras>
                        <obra>
                                <idObra>17</idObra>
                                …
```

La problemática es evidente, se entra en una recursividad infinita. JAXB no proporciona una solución para estas situaciones[38]. Algunas implementaciones de JAXB sí lo hacen (como EclipseLink MOXy[39]), pero en este proyecto hemos optado por utilizar tan sólo la API estándar que incluye Java SE[40].

Para atajar este problema, marcaremos las propiedades correspondientes a las relaciones entre clases con la anotación @XmlTransient en lugar de @XmlElement, de modo que se omitan de la serialización. Más adelante veremos cómo serializar esta información.

Por último, y relacionado con la serialización, añadiremos a las clases un método que sobreescriba el método *equals* de Object. Esto se debe a que debemos ser capaces de comparar dos objetos que representen la misma instancia de, por ejemplo, Autor. Este caso se puede dar al realizar el *unmarshalling* de un objeto que se nos pasa como parámetro en un servicio web y compararlo con una instancia cargada en el Entity Manager. Aunque representen el mismo Autor, por seguir con el ejemplo, al no ser la misma instancia de la clase Autor pueden surgir problemas, especialmente durante las operaciones de conjuntos. Con la sobreescritura del método por defecto solventaremos estos percances. Concretamente, la función *equals* comparará los identificadores de las instancias a comparar.

8.1.2.2.3. Clases generadas

Después de todas estas consideraciones, veamos el ejemplo de una clase generada por Hibernate Tools y modificada posteriormente atendiendo a lo antes expuesto:

```
package es.lta.biblioteca.model;
// Generated 06-may-2013 19:24:56 by Hibernate Tools 3.2.2.GA
```

---

[38] En la misma guía de referencia de JAXB se trata este tema. Lamentablemente, y más tratándose de una guía de referencia, no se da solución alguna para el caso de las relaciones Many-to-Many. https://jaxb.java.net/2.2.6/docs/ch03.html#annotating-your-classes-mapping-cyclic-references-to-xml

[39] http://www.eclipse.org/eclipselink/moxy.php

[40] Java Standard Edition

```
import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;


import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

/**
 * Obra generated by hbm2java
 */
@Entity
@Table(name="Obra"
    ,catalog="biblioteca"
)
@XmlRootElement(name="obra")
@XmlAccessorType(XmlAccessType.FIELD)
public class Obra  implements java.io.Serializable {

    @XmlElement private Integer idObra;
    @XmlElement private String titulo;
    @XmlElement private String metaInf;
    @XmlElement private String typeSettings;
    @XmlTransient private Set<Edicion> edicions = new HashSet<Edicion>(0);
    @XmlTransient private Set<Autor> autors = new HashSet<Autor>(0);
    @XmlTransient private Set<Etiqueta> etiquetas
                                          = new HashSet<Etiqueta>(0);

    public Obra() {
    }


    public Obra(String titulo) {
        this.titulo = titulo;
    }

    public Obra(String titulo, String metaInf, String typeSettings,
                Set<Edicion> edicions, Set<Autor> autors,
                Set<Etiqueta> etiquetas) {
       this.titulo = titulo;
       this.metaInf = metaInf;
       this.typeSettings = typeSettings;
       this.edicions = edicions;
       this.autors = autors;
```

```
   this.etiquetas = etiquetas;
}

@Id
@GeneratedValue(strategy=IDENTITY)
@Column(name="idObra", unique=true, nullable=false)
public Integer getIdObra() {
    return this.idObra;
}

public void setIdObra(Integer idObra) {
    this.idObra = idObra;
}

@Column(name="titulo", nullable=false, length=150)
public String getTitulo() {
    return this.titulo;
}

public void setTitulo(String titulo) {
    this.titulo = titulo;
}

@Column(name="metaInf", length=65535)
public String getMetaInf() {
    return this.metaInf;
}

public void setMetaInf(String metaInf) {
    this.metaInf = metaInf;
}

@Column(name="typeSettings", length=65535)
public String getTypeSettings() {
    return this.typeSettings;
}

public void setTypeSettings(String typeSettings) {
    this.typeSettings = typeSettings;
}

@ManyToMany(
  cascade =
     {CascadeType.PERSIST,CascadeType.REFRESH,CascadeType.MERGE},
  fetch=FetchType.EAGER)
@JoinTable(name="Obra_Edicion", catalog="biblioteca",
  joinColumns = {
     @JoinColumn(name="idObra", nullable=false, updatable=false) },
  inverseJoinColumns = {
     @JoinColumn(name="idEdicion", nullable=false, updatable=false)
})
public Set<Edicion> getEdicions() {
    return this.edicions;
}

public void setEdicions(Set<Edicion> edicions) {
    this.edicions = edicions;
```

```
    }

    @ManyToMany(
       cascade =
          {CascadeType.PERSIST,CascadeType.REFRESH,CascadeType.MERGE},
       fetch=FetchType.EAGER,
       mappedBy="obras")
    public Set<Autor> getAutors() {
        return this.autors;
    }

    public void setAutors(Set<Autor> autors) {
        this.autors = autors;
    }

    @ManyToMany(
       cascade =
          {CascadeType.PERSIST,CascadeType.REFRESH,CascadeType.MERGE},
       fetch=FetchType.EAGER)
    @JoinTable(name="Etiquetado", catalog="biblioteca",
       joinColumns = {
          @JoinColumn(name="idObra", nullable=false, updatable=false) },
       inverseJoinColumns = {
          @JoinColumn(name="idEtiqueta", nullable=false, updatable=false)
    })
    public Set<Etiqueta> getEtiquetas() {
        return this.etiquetas;
    }

    public void setEtiquetas(Set<Etiqueta> etiquetas) {
        this.etiquetas = etiquetas;
    }


       @Override
       public boolean equals(Object obj) {
              if(obj instanceof Obra){
                     return ((Obra) obj)
                            .getIdObra().equals(this.getIdObra());
              }
              return super.equals(obj);
       }
}
```

Todas las clases generadas las reuniremos en un JAR para su posterior inclusión como dependencia. Maven se encargará de esta empaquetación y de la gestión de las dependencias.

8.1.3. Clases DAO

Una vez generado el modelo del dominio, crearemos unas clases de acceso a los datos. Estas son las llamadas clases DAO.

Básicamente, serán las encargadas de realizar las operaciones CRUD. Para ello, instanciarán el EntityManager y el controlador de transacciones de JPA. A través del EntityManager realizarán la lectura y la escritura a base de datos dentro, siempre, de una transacción.

Será necesario crear un DAO por cada una de las clases presentes en el dominio, divididos en interfaz e implementación. Estas clases proveerán las mismas funciones, diferenciándose tan sólo por la clase del modelo que manejan. Por tanto, lo más adecuado es crear un DAO genérico que después extiendan el resto de DAOs.

He aquí la interfaz del DAO genérico que hemos creado:

```
package es.lta.biblioteca.service.dao;

import java.io.Serializable;
import java.util.Collection;
import java.util.List;

public interface GenericDao<T extends Serializable, ID extends
Serializable> {
	public void save(T t) throws Exception ;

	@SuppressWarnings("unchecked")
	public void save(T... ts) throws Exception;

	public void save(List<T> ts) throws Exception;

	public T load(ID id);

	@SuppressWarnings("unchecked")
	public List<T> load(ID... ids);

	public List<T> load(List<ID> ids);

	public List<T> loadAll();

	public void update(T t) throws Exception;

	@SuppressWarnings("unchecked")
	public void update(T... ts) throws Exception;

	public void update(Collection<T> ts) throws Exception;

	public void delete(T t) throws Exception;

	@SuppressWarnings("unchecked")
	public void delete(T... ts) throws Exception;

	public void delete(Collection<T> ts) throws Exception;

	public void deleteById(ID id) throws Exception;

	@SuppressWarnings("unchecked")
	public void deleteById(ID... ids) throws Exception;

	public void deleteById(Collection<ID> ids) throws Exception;
```

```
	public List<T> findByQuery(String query);


	public void flush() throws Exception;
}
```

Lo que sigue es su implementación, también genérica:

```
package es.lta.biblioteca.service.dao.impl;

import java.io.Serializable;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.annotation.Resource;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import javax.transaction.UserTransaction;

import es.lta.biblioteca.service.dao.GenericDao;

public abstract class GenericDaoImpl<T extends Serializable, ID extends
Serializable> implements GenericDao<T, ID> {

	@PersistenceContext
	protected EntityManager em;

	@Resource
	private UserTransaction utx;

	private Class<T> type;

	@SuppressWarnings({ "unchecked", "rawtypes" })
	public GenericDaoImpl() {
	  Type t = getClass().getGenericSuperclass();
	  ParameterizedType pt = (ParameterizedType) t;
	  type = (Class) pt.getActualTypeArguments()[0];
    }

	public void save(T t) throws Exception{
		try {
			utx.begin();
			this.em.persist(t);
			utx.commit();
		} catch (Exception e) {
			utx.rollback();
			throw e;
```

```
	}
}

@SuppressWarnings("unchecked")
public void save(T... ts) throws Exception {
	for (T t : ts) {
		save(t);
	}
}

public void save(List<T> ts) throws Exception {
	for (T t : ts) {
		save(t);
	}
}

public T load(ID id) {
	return this.em.find(type, id);
}

@SuppressWarnings("unchecked")
public List<T> load(ID... ids) {
	List<T> loadedTs = new ArrayList<T>();
	for (ID id : ids) {
		loadedTs.add(load(id));
	}
	return loadedTs;
}

public List<T> load(List<ID> ids) {
	List<T> loadedTs = new ArrayList<T>();
	for (ID id : ids) {
		loadedTs.add(load(id));
	}
	return loadedTs;
}

public List<T> loadAll() {
	CriteriaQuery<T> criteriaQuery =
		em.getCriteriaBuilder().createQuery(type);
	Root<T> t = criteriaQuery.from(type);
	criteriaQuery.select(t);
	TypedQuery<T> typedQuery = em.createQuery(criteriaQuery);
	return typedQuery.getResultList();
}

public void update(T t) throws Exception{
	try {
		utx.begin();
		this.em.merge(t);
		utx.commit();
	} catch (Exception e) {
		utx.rollback();
		throw e;
	}
}
```

```
@SuppressWarnings("unchecked")
public void update(T... ts) throws Exception {
	for (T t : ts) {
		update(t);
	}

}

public void update(Collection<T> ts) throws Exception {
	for (T t : ts) {
		update(t);
	}

}

public void delete(T t) throws Exception {
	try {
		utx.begin();
		this.em.remove(t);
		utx.commit();
	} catch (Exception e) {
		utx.rollback();
		throw e;
	}
}

@SuppressWarnings("unchecked")
public void delete(T... ts) throws Exception {
	for (T t : ts) {
		delete(t);
	}

}

public void delete(Collection<T> ts) throws Exception {
	for (T t : ts) {
		delete(t);
	}

}

public void deleteById(ID id) throws Exception{
	try {
		utx.begin();
		this.em.remove(load(id));
		utx.commit();
	} catch (Exception e) {
		utx.rollback();
		throw e;
	}
}

@SuppressWarnings("unchecked")
public void deleteById(ID... ids) throws Exception {
	for (ID id : ids) {
		deleteById(id);
	}
```

```
	}

	public void deleteById(Collection<ID> ids) throws Exception {
		for (ID id : ids) {
			deleteById(id);
		}

	}

	public List<T> findByQuery(String query){
		return this.em.createQuery(query, type).getResultList();
	}

	public void flush() throws Exception {
		utx.begin();
		this.em.flush();
		utx.commit();
	}
}
```

Como vemos, las clases genéricas presentes en el DAO genérico son dos. Una es la clase del modelo para la que construir el DAO, y la otra el tipo de su identificador, ya que los métodos load y deleteById lo reciben como parámetro.

Definido este DAO genérico, bastará con extender para cada clase tanto la interfaz como la implementación. Este proceso lo haremos de nuevo con Hibernate Tools. Crearemos unas nuevas plantillas que se ejecuten para cada una de las tablas que encuentre al conectarse a nuestra base de datos. Estas plantillas son las siguientes:

```
<#if !pojo.isComponent()>
${pojo.getPackageDeclaration()}
// Generated ${date} by Hibernate Tools ${version}

<#include "extraImports.ftl">

<#foreach property in pojo.getAllPropertiesIterator()>
<#if property.equals(clazz.identifierProperty)>
public interface ${pojo.getDeclarationName()}Dao extends
GenericDao<${pojo.getDeclarationName()},
${pojo.getJavaTypeName(property, jdk5)}>{

}
</#if>
</#foreach>
</#if>
```

La de arriba corresponde a la generación de las interfaces. La plantilla extraImports.ftl referenciada incluye los imports necesarios para las clases. Para las implementaciones se utiliza la siguiente plantilla:

```
<#if !pojo.isComponent()>
${pojo.getPackageDeclaration()}
```

```
// Generated ${date} by Hibernate Tools ${version}

<#include "extraImports.ftl">

<#foreach property in pojo.getAllPropertiesIterator()>
<#if property.equals(clazz.identifierProperty)>
@Named("${pojo.getDeclarationName()?uncap_first}Dao")
public class ${pojo.getDeclarationName()}DaoImpl extends
GenericDaoImpl<${pojo.getDeclarationName()},
${pojo.getJavaTypeName(property, jdk5)}> implements
${pojo.getDeclarationName()}Dao{

}
</#if>
</#foreach>
</#if>
```

Al ejecutar la tarea de generación de DAOs se creará una clase de este tipo por cada una de las clases del dominio. Veamos por ejemplo cómo queda para el caso de la clase Etiqueta:

```
package es.lta.biblioteca.service.dao;
// Generated 04-may-2013 2:27:37 by Hibernate Tools 3.2.2.GA

import es.lta.biblioteca.model.*;
import es.lta.biblioteca.service.dao.GenericDao;

public interface EtiquetaDao extends GenericDao<Etiqueta, Integer>{

}
```

Y la implementación de esta interfaz:

```
package es.lta.biblioteca.service.dao.impl;
// Generated 04-may-2013 2:27:37 by Hibernate Tools 3.2.2.GA

import es.lta.biblioteca.model.*;
import es.lta.biblioteca.service.dao.EtiquetaDao;

import javax.inject.Named;

@Named("etiquetaDao")
public class EtiquetaDaoImpl extends GenericDaoImpl<Etiqueta, Integer>
implements EtiquetaDao{

}
```

Todos los DAOs generados los empaquetaremos en un mismo JAR con una dependencia que nos gestionará Maven al modelo del dominio.

8.1.3.1. Inyección de dependencias

Hay algunos aspectos de estos DAOs que hemos dejado de lado hasta ahora y que resultan fundamentales para el funcionamiento de los mismos. Se trata de las anotaciones y configuraciones que permitirán inyectar dependencias.

Fijémonos en primer lugar en la clase GenericDaoImpl, más concretamente en las anotaciones @PersistenceContext y @Resource.

```
@PersistenceContext
protected EntityManager em;

@Resource
private UserTransaction utx;
```

Estas anotaciones están indicando que, en el momento de instanciación de la clase, se debe inyectar el EntityManager y el controlador de transacciones de JPA. Para que en tiempo de ejecución la JVM sepa que se está realizando un mapeo objeto relacional con JPA, y conozca qué EntityManager y controlador de transacciones ha de inyectar en estas clases, deberemos incluir en la ruta META-INF del JAR con los DAOs un fichero de nombre persistence.xml. En nuestro caso es el siguiente:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation=
		"http://java.sun.com/xml/ns/persistence
		http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
	version="2.0">
	<persistence-unit name="es.lta.biblioteca">
		<jta-data-source>es.lta.biblioteca</jta-data-source>
	</persistence-unit>
</persistence>
```

donde es.lta.datasource es un datasource que habremos configurado externamente (lo veremos cuando tratemos del servidor).

Estas inyecciones de dependencias forman parte del método de trabajar con JPA. Lo siguiente que veremos no está relacionado con JPA, pero sí con la inyección de dependencias.

8.1.3.1.1. CDI

Se trata de la anotación @Named de las clases xxxDaoImpl. Esta anotación pertenece a la API de Java EE para la inyección de dependencias y contextos, CDI[41].

Con esta anotación, estamos diciendo que queremos que se instancie la clase cuando se cargue en la JVM, y que esa instancia se ofrezca a aquellas clases que desean inyectarla como dependencia.

De este modo, tenemos cada uno de los DAOs ya instanciados y listos para ser inyectados en las clases que los requieran.

[41] *Contexts and Dependency Injection*, es el JSR 299, http://www.jcp.org/en/jsr/detail?id=299

Para activar la gestión de dependencias por CDI es necesario que, junto al fichero persistence.xml, incluyamos en la ruta META-INF el siguiente fichero beans.xml:

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">

</beans>
```

En nuestro caso el fichero es así de simple, ya que todos los beans están expuestos haciendo uso de las anotaciones de CDI directamente en las clases.

Spring[42] es una alternativa al uso de CDI, y es también una implementación de esta API. Sin embargo, ya dijimos que primaríamos el uso de APIs de JavaEE y, además, la funcionalidad de inyección de dependencias que utilizaremos será muy básica.

8.1.4. Servicios

Para acabar con el módulo de datos, por encima de los DAOs crearemos unas clases de servicio. Una para cada entidad.

Estas clases proveerán métodos CRUD y funciones para establecer las relaciones entre clases. En este punto, y de acuerdo con el alcance del proyecto, hemos implementado las clases de servicio para Autor y Obra, y los hemos empaquetado en un JAR con sus dependencias a los DAOs y el modelo del dominio.

Veamos la interfaz AutorService:

```
package es.lta.biblioteca.service;

import …

public interface AutorService {

       public Autor load(Integer id);

       public List<Autor> loadAll();

       public Autor saveOrUpdate(Autor autor);

       public void delete(Integer id) throws Exception;

       public void setObras(Integer id, Collection<Obra> obras);

       public void setParticipaciones(
```

[42] http://www.springsource.org/, se trata de un completísimo framework de desarrollo de aplicaciones. Aquí nos estamos refiriendo a su faceta como gestor de dependencias, por la que es originalmente conocido

```
            Integer id, Collection<Participacion> participaciones);
}
```

Hemos omitido los comentarios de la clase para ahorrar espacio, asi como los imports. Las funciones para establecer relaciones se reflejan aquí en la presencia de las funciones setObras y setParticipaciones.

Veamos ahora por encima la implementación de esta interfaz:

```
package es.lta.biblioteca.service.impl;

import javax.inject.Inject;
import javax.inject.Named;
import …

@Named("autorService")
public class AutorServiceImpl implements AutorService{

	@Inject
	AutorDao autorDao;

	@Inject
	ObraDao obraDao;

	@Inject
	ParticipacionDao participacionDao;
```

Aquí, de nuevo, estamos indicando con @Named que la clase se instanciará para que sea inyectada en las clases que la pidan. A su vez, en el momento de la instanciación se están requiriendo instancias de tres DAOs, los correspondientes a la clase Autor, Obra y Participacion. Estos DAOs estarán ya instanciados porque así lo indicamos con @Named en aquellas clases, y serán inyectados aquí.

Continuemos viendo la implementación que se le da al método saveOrUpdate:

```
	public Autor saveOrUpdate(Autor autor) {
		Autor autorOld = null;
		if(autor.getIdAutor()!=null){
			autorOld = autorDao.load(autor.getIdAutor());
		}
		if(autorOld!=null){
			autor.setObras(autorOld.getObras());

	autor.setParticipaciones(autorOld.getParticipaciones());
			try{
				autorDao.update(autor);
			}catch(Exception e){
				throw new WebApplicationException(e);
			}
		}else{
			try{
				autor.setIdAutor(null);
```

```
                autorDao.save(autor);
            }catch(Exception e){
                throw new WebApplicationException(e);
            }
        }
        return autor;
    }
```

El método consultará primero si al autor existe. Si no existe, lo creará. Si existe, lo actualizará. Todo ello lo hace a través de los métodos definidos por el DAO.

Cabe destacar que para preservar la coherencia en las relaciones, el método saveOrUpdate hace caso omiso de las relaciones que contenga la instancia recibida como parámetro. En el ejemplo de arriba, no se guardan los cambios en las obras o participaciones. Para ello han de utilizarse los métodos setObras y setParticipaciones. Veamos el método setObras a modo de ejemplo:

```
    public void setObras(Integer id, Collection<Obra> obras) {
        Autor autor = autorDao.load(id);
        if(autor == null){
            throw new WebServiceException("No existe el autor.");
        }
        List<Obra> obrasRecovered = new ArrayList<Obra>();
        for(Obra obra : obras){
            obrasRecovered.add(obraDao.load(obra.getIdObra()));
        }
        obras=obrasRecovered;

        List<Obra> toRemove = new ArrayList<Obra>(autor.getObras());
        List<Obra> toAdd = new ArrayList<Obra>(obras);
        List<Obra> intersection = new ArrayList<Obra>(toRemove);
        intersection.retainAll(obras);
        toRemove.removeAll(intersection);
        toAdd.removeAll(intersection);

        autor.setObras(new HashSet<Obra>(obras));
        try {
            autorDao.update(autor);
        } catch (Exception e) {
            throw new WebApplicationException(e);
        }

        for(Obra obra : toAdd){
            obra.getAutors().add(autor);
            try {
                obraDao.update(obra);
            } catch (Exception e) {
                throw new WebApplicationException(e);
            }
        }

        for(Obra obra : toRemove){
            obra.getAutors().remove(autor);
            try {
                obraDao.update(obra);
```

```
                } catch (Exception e) {
                        throw new WebApplicationException(e);
                }
        }
}
```

Todo este procedimiento está destinado a preservar la correspondencia entre las instancias de las clases[43].

Ésta ha sido la solución adoptada en este proyecto. Sin duda, adoptando otros frameworks (ya entraremos en ello más adelante) o metodologías, podríamos evitarlo. Incluso podría evolucionarse esta solución, añadiendo complejidad al método saveOrUpdate(). Pero por el momento, bastará con conocer las restricciones de estos métodos.

8.2. Módulo de servicios web

Una vez implementado el núcleo de nuestra aplicación, que lo conforma el módulo de datos, pasamos a implementar los servicios web que expondremos. Como ya dijimos anteriormente, estos servicios web serán de tipo REST.

Antes de implementar los servicios REST debemos estudiar los recursos que se expondrán. Esto es, deberemos identificar la URI[44] de cada uno, el método HTTP que se utilizará para su acceso y el formato de los mensajes tanto de entrada como de salida. Veamos el estudio correspondiente a la parte de implementación del sistema que estamos cubriendo en este proyecto.

A la hora de exponer este estudio indicaremos la URI del recurso expuesto (que una vez desplegados los servicios web formará parte de la URL con la que hay que acceder al recurso), el método HTTP que hay que utilizar, qué entrada de datos se requiere en el cuerpo del mensaje en la *request* (consume) y qué salida de datos se produce en la *response* (produce). No se indica en cada caso, pero todas estas entradas y salidas serán en formato XML o JSON, de forma indistinta. Los elementos comprendidos entre llaves en la URI constituyen variables de entrada.

8.2.1. Recursos expuestos para los autores

Los servicios expuestos permitirán realizar el mantenimiento de los autores. Esto es, crearlos, modificarlos, borrarlos y consultarlos.

URI: /autor/{id}
HTTP: GET
Produce: Datos del autor con identificador *id*
Devuelve los datos del autor con identificador *id* solicitado

---

[43] Puede encontrarse más información sobre esta preservación de la correspondencia en http://en.wikibooks.org/wiki/Java_Persistence/Relationships#Object_corruption.2C_one_side_of_the_relationship_is_not_updated_after_updating_the_other_side
[44] *Uniform Resource Identifier*, Identificador Uniforme de Recursos

URI: /autor
HTTP: GET
Produce: Datos de todos los autores en el sistema
Devuelve los datos de todos los autores

URI: /autor
HTTP: POST
Consume: Datos de un autor
Produce: Datos del autor actualizados
Actualiza un autor existente o crea uno nuevo si no existe

URI: /autor/{id}
HTTP: DELETE
Borra el autor con identificador *id* indicado

URI: /autor/{id}/obras
HTTP:GET
Produce: Datos de las obras del autor con identificador *id*
Devuelve las obras del autor con identificador *id*

URI: /autor/{id}/obras
HTTP:POST
Consume: Datos de las obras
Actualiza las autorías del autor con identificador *id*

URI: /autor/{id}/participaciones
HTTP:GET
Produce: Datos de las participaciones del autor con identificador *id*
Devuelve las participaciones del autor con identificador *id*

URI: /autor/{id}/ participaciones
HTTP:POST
Consume: Datos de las participaciones
Actualiza las participaciones del autor con identificador *id*

8.2.2. Recursos expuestos para las obras

En el caso de las obras los recursos expuestos permitirán operaciones análogas a las vistas antes.

URI: /obra/{id}
HTTP: GET
Produce: Datos de la obra con identificador *id*
Devuelve los datos de la obra con identificador *id* solicitada

URI: /obra
HTTP: GET
Produce: Datos de todas las obras en el sistema
Devuelve los datos de todas las obras

URI: /obra
HTTP: POST
Consume: Datos de una obra
Produce: Datos de la obra actualizados
Actualiza una obra existente o crea una nueva si no existe

URI: /obra/{id}
HTTP: DELETE
Borra la obra con identificador *id* indicado

URI: /obra/{id}/ediciones
HTTP:GET
Produce: Datos de las ediciones de la obra con identificador *id*
Devuelve las ediciones de la obra con identificador *id*

URI: /obra/{id}/ediciones
HTTP:POST
Consume: Datos de las ediciones
Actualiza las ediciones asociadas de la obra con identificador *id*

URI: /obra/{id}/autores
HTTP:GET
Produce: Datos de los autores de la obra con identificador *id*
Devuelve los autores de la obra con identificador *id*

URI: /obra/{id}/ autores
HTTP:POST
Consume: Datos de los autores
Actualiza los autores de la obra con identificador *id*

URI: /obra/{id}/etiquetas
HTTP:GET
Produce: Datos de las etiquetas de la obra con identificador *id*
Devuelve las etiquetas de la obra con identificador *id*

URI: /obra/{id}/ etiquetas
HTTP:POST
Consume: Datos de las etiquetas
Actualiza las etiquetas asignadas a la obra con identificador *id*

8.2.3. Marco de desarrollo

Siguiendo la tónica presente hasta ahora, para la implementación de los servicios optaremos por el estándar definido por Java EE en su versión 6, dejando al servidor web en

el que despleguemos los servicios el uso de una implementación u otra. La API dedicada a los servicios REST de Java EE es JAX-RS[45].

Las implementaciones open source más destacadas de esta API son RESTEasy[46], CXF[47] y Jersey[48]. Cada una de ellas expande la API de forma significativa, especialmente las dos primeras. Volveremos a hablar de ellas en algunos de los puntos siguientes.

8.2.4. Implementación de los servicios REST

Para la implementación de los servicios bastará con escribir las clases con los métodos correspondientes a los recursos expuestos arriba y utilizar las anotaciones que provee la API. Veamos el ejemplo de la clase programada para exponer los recursos que antes hemos identificado para el autor:

```
package es.lta.biblioteca.service.rest.impl;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.MediaType;

import es.lta.biblioteca.model.Autor;
import es.lta.biblioteca.model.Obra;
import es.lta.biblioteca.model.Participacion;
import es.lta.biblioteca.model.helper.Autores;
import es.lta.biblioteca.model.helper.Obras;
import es.lta.biblioteca.model.helper.Participaciones;
import es.lta.biblioteca.service.AutorService;
import es.lta.biblioteca.service.rest.AutorServiceREST;

@Path("/autor")
public class AutorServiceRESTImpl implements AutorServiceREST {

	@Inject
```

[45] *Java API for RESTful Web Services*, es el JSR 311, http://www.jcp.org/en/jsr/detail?id=311. En este proyecto se utiliza la versión 1.1 incluida en el perfil completo de Java EE 6.
[46] Proyecto de JBoss, http://www.jboss.org/resteasy.
[47] http://cxf.apache.org/, parte de la familia Apache. No sólo se trata de una implementación de JAX-RS, sino que también lo es de JAX-WS (*Java API for XML-Based Web Services*, JSR 224, http://www.jcp.org/en/jsr/detail?id=224).
[48] Se trata de la implementación de referencia de la API, https://jersey.java.net/.

```
AutorService autorService;

@GET
@Path("{id}")
@Produces(
	{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public Autor load(@PathParam("id") Integer id) {
	return autorService.load(id);
}

@GET
@Produces(
	{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public List<Autor> loadAll() {
	return new Autores(autorService.loadAll()).getAutores();
}

@POST
@Consumes(
	{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
@Produces(
	{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public Autor saveOrUpdate(Autor autor) {
	return autorService.saveOrUpdate(autor);
}

@DELETE
@Path("{id}")
public void delete(@PathParam("id") Integer id) {
	try {
		autorService.delete(id);
	} catch (Exception e) {
		throw new WebApplicationException(e);
	}
}

@GET
@Path("{id}/obras")
@Produces(
	{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public List<Obra> getObras(@PathParam("id") Integer id){
	Autor autor = autorService.load(id);
	if(autor==null){
		return null;
	}
	return new Obras(new ArrayList<Obra>(autor.getObras())
			).getObras();
}

@POST
@Path("{id}/obras")
@Consumes(
	{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public void setObras(
	@PathParam("id") Integer id, Collection<Obra> obras){
	autorService.setObras(id, obras);
}
```

```
	@GET
	@Path("{id}/participaciones")
	@Produces(
		{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public List<Participacion> getParticipaciones(
					@PathParam("id") Integer id){
		Autor autor = autorService.load(id);
		if(autor==null){
			return null;
		}
		return new Participaciones(
			new ArrayList<Participacion>(
				autor.getParticipaciones()))
				.getParticipaciones();
	}

	@POST
	@Path("{id}/participaciones")
	@Consumes(
		{ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
	public void setParticipaciones(
			@PathParam("id") Integer id,
			Collection<Participacion> participaciones){
		autorService.setParticipaciones(id, participaciones);
	}
}
```

Fijémonos en primer lugar que estamos inyectando el servicio AutorService que hemos creado. Esta clase es la que nos proporciona toda la lógica, haciendo nosotros en esta clase un mínimo tratado de los datos en algunos casos.

Pasemos a tratar las anotaciones de JAX-RS que dotan a la clase de la funcionalidad que buscamos.

En primer lugar encontramos la anotación @Path("/autor"), que nos indica que todos los recursos se despliegan en el contexto "/autor" de la aplicación REST a la que pertenezcan.

Para cada uno de los métodos expuestos como servicios hemos de indicar el método al que responden. Para ello usamos aquí las anotaciones @GET, @POST y @DELETE.

Aquellos recursos que presentan en su URI más información que la del contexto (/autor), van anotadas con @Path(…) para ampliar esta URI. Las variables de entrada se incluyen aquí entre llaves y después podemos referenciarlas con la anotación @PathParam al declarar los parámetros del método.

Por último, las anotaciones @Consumes y @Produces indican los MIME-Type que aceptan y producen, respectivamente, los métodos. En nuestro caso hemos indicado que los tipos son XML y JSON. No hemos de preocuparnos por recuperar manualmente el texto del cuerpo de la *request* o escribir en la *response* el texto con la información que se devuelve. JAX-RS tiene soporte nativo de JAXB, con lo que se realiza el *marshalling* y *unmarshalling* correspondiente a las clases que indicamos como entradas y salidas.

De este modo, el Autor que se devuelve en el método *load* se serializa automáticamente en XML en la *response*, y el Autor que se recoge de la *request* en formato XML en el método saveOrUpdate se deserializa también automáticamente.

Lo mismo ocurre con el formato JSON, pero no gracias a JAX-RS sino a sus implementaciones. La mayoría de ellas incluyen esta serialización y deserialización automática para JSON haciendo uso de soluciones como Jackson[49] o Jettison[50], que reutilizan las anotaciones JAXB.

Relacionado con JAXB, hemos tenido que adoptar un *workaround* para solventar un comportamiento no esperado relacionado con la serialización de las colecciones que contienen las relaciones de las entidades. A la hora de hacerlo, se serializaban del siguiente modo:

<obras>1, 4, 5, 7 <obras>

que no constituye el comportamiento deseado y provocaba errores en el momento de la deserialización. Por tanto, nos hemos valido de unas clases auxiliares. Podemos verlo reflejado en el método getObras, por ejemplo. En este caso utilizamos la clase auxiliar Obras:

```
package es.lta.biblioteca.model.helper;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;

import es.lta.biblioteca.model.Obra;

public class Obras {

	@XmlElementWrapper
	@XmlElement(name="obra")
	private List<Obra> obras;

	public List<Obra> getObras() {
		return obras;
	}

	public void setObras(List<Obra> obras) {
		this.obras = obras;
	}

	public Obras(List<Obra> obras) {
		super();
		this.setObras(obras);
```

[49] http://jackson.codehaus.org/
[50] http://jettison.codehaus.org/

```
        }
}
```

Usando esta clase auxiliar los errores quedaron solventados.

Por último, nótese que se ha creado una interfaz para las clases correspondientes a los servicios web. Se trata de interfaces con los mismos métodos y las mismas anotaciones para ellos. Estas interfaces anotadas nos serán de utilidad a la hora de construir el cliente de los servicios REST, como veremos más adelante.

La razón por la que tanto la interfaz como la implementación tengan que estar anotadas con las mismas anotaciones JAX-RS es que no todas las implementaciones de JAX-RS permiten que sólo la interfaz lo esté. Si sólo anotáramos la interfaz veríamos que nuestro servicio se publica sólo con algunas implementaciones, mientras que con otras no.

A la hora de empaquetar estas clases con los servicios web y las clases auxiliares lo haremos esta vez en un WAR, que desplegaremos, como veremos a continuación, en un servidor Java. A su vez, las interfaces generadas las empaquetamos aparte como JAR, para poder utilizarlas más adelante separadas de la implementación.

El WAR contendrá todos los JARs de los que depende. Maven se encarga de todo el proceso de empaquetado y búsqueda de dependencias.

Ya que se trata de una aplicación web que desplegaremos en un servidor, debemos asegurarnos de incluir el fichero web.xml. En nuestro caso, como desplegaremos el WAR en un servidor Java EE, bastará con crear el siguiente fichero.

```
<?xml version="1.0"?>
<web-app>

</web-app>
```

Todas las demás configuraciones se harán a través de las anotaciones encontradas en el código y los ficheros incluidos en las carpetas META-INF del WAR y las librerías JAR de las dependencias incluidas.

8.3. Servidor Java EE para los servicios web

Una vez tenemos el WAR con los servicios web listos para desplegar necesitamos un servidor donde hacerlo. Para ello, buscamos un servidor certificado para Java EE 6 que nos provea de las implementaciones de las APIs que hemos utilizado. El servidor lo instalaremos en la máquina de desarrollo, no en la máquina virtual, para mayor comodidad a la hora de realizar los despliegues.

8.3.1. JBoss AS

Nuestra primera opción ha sido JBoss AS[51], concretamente la versión 7.1.1 Final, que está certificada para el perfil completo de Java EE 6. Las implementaciones que ofrece JBoss AS para las tecnologías que hemos adoptado en nuestro proyecto son:

JPA	Hibernate
JAXB	La provista por Java SE y varias incluidas en RESTEasy. Cada una es utilizada según el caso.
CDI	Weld
JAX-RS	RESTEasy

Todas las implementaciones, excepto las de JAXB, son de la propia JBoss (que, aunque no lo hemos citado antes, pertenece a Red Hat).

Tras haber configurado el *datasource* a través de su interfaz gráfica, desplegamos el WAR con los sericios web en el servidor. JBoss registra la aplicación web pero no se expone ningún servicio. Tras haber hecho multitud de pruebas y de ajustes extras a lo estrictamente necesario que marca JAX-RS, no hemos conseguido que los servicios se expongan en JBoss, por lo que desechamos el servidor y buscamos otro que no nos cause tantos problemas.

8.3.2. Apache TomEE

El siguiente candidato es Apache TomEE[52]. Se trata de un servidor open source certificado para el perfil web de Java EE 6 y construido sobre Tomcat[53]. El perfil web de Java EE 6 no incluye JAX-RS, pero Apache ofrece una versión extendida de TomEE que incluye algunas especificaciones añadidas a este perfil. Se trata de TomEE+. En nuestro caso la versión 1.5.1. Las implementaciones para las tecnologías utilizadas son, en este caso:

JPA	OpenJPA
JAXB	La provista por Java SE y varias incluidas en CXF. Cada una es utilizada según el caso.
CDI	OpenWebBeans
JAX-RS	CXF

Todas las implementaciones, exceptuando las de JAXB, son de Apache.

Configuramos el datasource editando el fichero $TOMEE_HOME/conf/tomee.xml, donde $TOMEE_HOME es el directorio raíz del servidor, donde hemos descomprimido el fichero que nos hemos descargado de la web de Apache TomEE.

---

[51] http://www.jboss.org/jbossas, servidor Java open source de JBoss. Recientemente ha sido anunciado que será renombrado a WildFly, http://wildfly.org/.
[52] http://tomee.apache.org/, de Apache.
[53] Contenedor de Servlets y JSPs de Apache, http://tomcat.apache.org/

La definición que hemos añadido al fichero tomee.xml es la siguiente:

```
<Resource id="es.lta.biblioteca" type="DataSource">
JdbcDriver  com.mysql.jdbc.Driver
JdbcUrl jdbc:mysql://vmPfc/biblioteca
UserName    biblioteca
Password     biblioteca
JtaManaged true
</Resource>
```

Después copiaremos a la carpeta $TOMEE_HOME/webapps el WAR con nuestros servicios web que, por cierto, se llama biblioteca-rest-services.war. Al arrancar el servidor (con $TOMEE_HOME/bin/startup.bin) se despliega la aplicación y queda disponible en el contexto / biblioteca-rest-services.

Hacemos una prueba ingresando en un navegador la siguiente URL:

http://localhost:8080/biblioteca-rest-services/autor/1

Y obtenemos la siguiente respuesta:

```
<autor>
      <idAutor>1</idAutor>
      <nombre>Miguel</nombre>
      <apellido1>Cervantes, de</apellido1>
      <apellido2>Saavedra</apellido2>
      <imagenUrl>
      http://upload.wikimedia.org/wikipedia/commons/6/66/Cervates_j
      auregui.jpg
      </imagenUrl>
</autor>
```

Intentamos ahora obtener la respuesta en formato JSON. Para ello utilizamos una aplicación para Google Chrome llamada Postman – REST Client[54], en la que podemos fijar el parámetro Accept del *header* de la *request* a application/json. Realizamos, pues, la *request* de tipo GET a la misma URL de antes y obtenemos lo siguiente:

```
{
    "autor": {
        "idAutor": 1,
        "nombre": "Miguel",
        "apellido1": "Cervantes, de",
        "apellido2": "Saavedra",
        "imagenUrl":
         "http://upload.wikimedia.org/wikipedia/commons/6/66/Cervat
         es_jauregui.jpg"
    }
}
```

[54] https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm

Así pues, esta vez sí que se ha desplegado correctamente nuestra aplicación y podemos pasar a tratar sobre las interfaces gráficas.

8.4. Módulo de interfaz gráfica

Con nuestros servicios web en marcha sólo falta crear las interfaces con las que interactuará el usuario. Recordemos que hemos planteado dos modos de llevar a cabo esa interacción: a través de una navegador web desde un pc corriente, y haciendo uso de una aplicación nativa para Android. Comencemos por la aplicación web.

8.4.1. Aplicación web

En un primer momento hemos considerado implementar la aplicación web haciendo uso de una herramienta que automatiza la creación de interfaces de administración. En concreto, se trata de Forge[55], de JBoss.

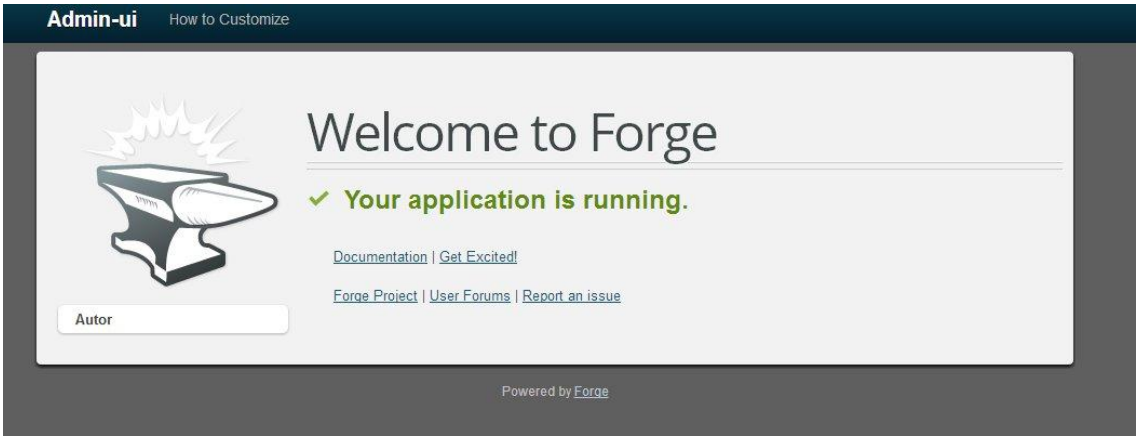
8.4.1.1. Forge

Forge es una herramienta RAD[56], entre cuyas capacidades se encuentra la de crear una aplicación de mantenimiento a partir de clases de dominio anotadas en JPA. Se trata de una herramienta de línea de comandos.

De forma abreviada, arrancamos Forge, le pedimos que cree un nuevo proyecto Maven indicando que será para una aplicación generada a partir de entidades JPA, señalamos dónde se encuentran nuestras entidades anotadas y Forge creará una aplicación JSF[57]. Esta técnica se denomina *scaffolding*[58]. Esta aplicación JSF consta de una vista general con un enlace a la administración de cada una de las entidades, y con tres vistas por cada entidad: una de creación, otra de detalle y una última para la búsqueda. La parte visual de la aplicación recibe los estilos de Bootstrap[59].

---

[55] http://forge.jboss.org/
[56] *Rapid Application Development*, Desarrollo Rápido de Aplicaciones.
[57] Java Server Faces
[58] *Scaffold* significa en inglés “andamio”.
[59] http://twitter.github.io/bootstrap/, se trata de un framework CSS y JavaScript para el front-end de las aplicaciones web.



Página principal de la aplicación generada por Forge

Los *backing beans* que Forge ha creado para estas tres vistas contienen toda la lógica de la administración de la entidad administrada. Se trata de clases que utilizan las APIs de JPA, EJB y JSF. Las entidades se escriben y se consultan desde un *entity manager* creado por la aplicación.

Como estos *backing beans* no utilizan nuestros servicios web para la explotación de los datos nos encontramos que Forge tan sólo nos libera de parte del trabajo, ya que es necesario revisar toda la lógica para hacer desaparecer toda la parte JPA de la misma sustituyéndola por llamadas a los servicios web. Por tanto, descartamos continuar por esta línea.

Otra herramienta RAD, más extendida en este caso, es Spring ROO[60]. Sin embargo, mientras que Forge genera código "agnóstico" (en el sentido de utilizar APIs estándar definidas por Java EE), ROO está ligado al ecosistema Spring. Además, mucho nos tememos que la aplicación que genere ROO también tendrá que ser considerablemente modificada.

Por tanto, decidimos implementar la interfaz desde cero.

8.4.1.2. Vaadin

Para construir la aplicación hemos optado por utilizar el *framework* Vaadin[61]. Se trata de un *framework* de desarrollo de aplicaciones RIA[62] basado en GWT[63] y HTML5. Con él, el desarrollador programa la interfaz en Java, describiendo el comportamiento que ha de seguir. Más tarde, el compilador del *framework* generará el código destinado al cliente web. Esto es, traducirá las clases Java que hemos escrito antes a HTML y Javascript, creando una aplicación AJAX[64] con el comportamiento que antes hemos programado.

---

[60] http://www.springsource.org/spring-roo
[61] https://vaadin.com/home
[62] *Rich Interface Applications*
[63] *Google Web Toolkit*, https://developers.google.com/web-toolkit/.
[64] *Asynchronous JavaScript And XML*

El arquetipo para Maven que proveen los propios desarrolladores de Vaadin nos permite generar la estructura básica del proyecto.

La forma de definir esta interfaz es muy similar a la conocida por AWT o SWING[65]: declararemos los componentes que aparecen en pantalla y los comportamientos que han de seguir mediante *listeners*, eventos y clases anónimas.

Veamos, como primer ejemplo ilustrativo, la clase que define la vista principal de la aplicación:

```
package es.lta.biblioteca.admin;

import …

import es.lta.biblioteca.admin.components.EditorInterface;
import es.lta.biblioteca.admin.components.impl.AutorEditor;
import es.lta.biblioteca.admin.components.impl.ObraEditor;

/**
 * The Application's "main" class
 */
@SuppressWarnings("serial")
@Theme("bibliotecatheme")
public class AdminUI extends UI {

	@Override
	protected void init(VaadinRequest request) {
		final HorizontalSplitPanel horizontalSplitPanel
			= new HorizontalSplitPanel();
		horizontalSplitPanel.setSizeFull();
		horizontalSplitPanel.setSplitPosition(
					120, Sizeable.Unit.PIXELS);
		setContent(horizontalSplitPanel);

		Tree tree = new Tree();
		horizontalSplitPanel.addComponent(tree);
		tree.setSelectable(true);
		tree.setNullSelectionAllowed(false);
		tree.setImmediate(true);

		final Map<String,AbsoluteLayout> treeItems
			= new LinkedHashMap<String, AbsoluteLayout>();
		treeItems.put("Autores", new AutorEditor());
		treeItems.put("Obras", new ObraEditor());
		for(Entry<String, AbsoluteLayout> entry :
					treeItems.entrySet())
		{
			tree.addItem(entry.getValue());
			tree.setItemCaption(entry.getValue(), entry.getKey());
			tree.setChildrenAllowed(entry.getValue(), false);
		}
```

[65] AWT y SWING son librerías gráficas presentes en Java SE. La segunda extiende la primera.

```
                tree.addValueChangeListener(
                                new Property.ValueChangeListener() {

                        @SuppressWarnings("unchecked")
                        @Override
                        public void valueChange(ValueChangeEvent event) {
                                Property<AbsoluteLayout> prop
                                        = event.getProperty();
                                ((EditorInterface)prop.getValue()).reView();
                                horizontalSplitPanel
                                        .setSecondComponent(prop.getValue());
                        }
                });

                // select first entity view
                tree.setValue(tree.getItemIds().iterator().next());
        }

}
```

Al acceder a la aplicación se ejecutará el método *init* de esta clase. En él declaramos un primer componente HorizontalSplitPanel, esto es, un panel dividido horizontalmente. Dentro, como primer componente, incluimos un listado con dos elementos, Autores y Obras, componentes que hemos creado y que nos permitirán la consulta y edición de las respectivas entidades. En el segundo componente del panel horizontal mostraremos estos editores en función del elemento del listado que escojamos, evento que hemos controlado con el método addValueChangeListener del Tree que constituye el listado. Por último, seleccionamos el primer elemento del listado por defecto.

La interfaz resultante es la siguiente:

Fragmento de pantalla de inicio de la aplicación web

Los componentes que se mostrarán a la derecha los hemos implementado en las clases AutorEditor y ObraEditor. Ambas clases extienden el componente de Vaadin AbsoluteLayout. La programación de estas clases las ilustraremos con fragmentos del código de AutorObra.

En primer lugar declaramos un constructor para la clase:

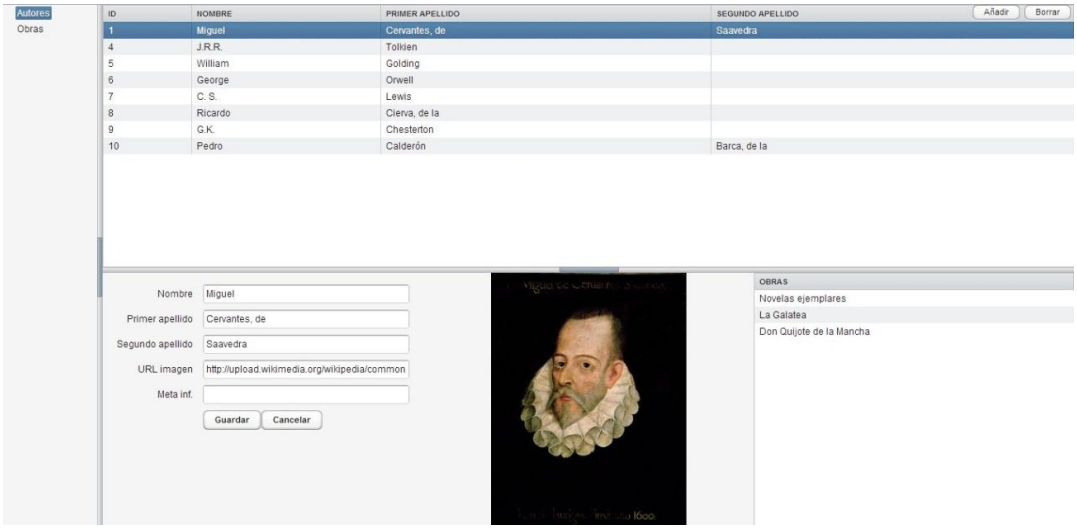
```
        public AutorEditor() {
                initFormFields();
                setSizeFull();
                initContainer();
```

```
                buildView();
        }
```

Es un constructor que ejecuta cuatro métodos. Los tres primeros inicializan una serie de parámetros que se utilizarán después, como los componentes internos del editor (botones, tablas, etc.) y otras variables. El método buildView es el encargado de definir la posición de esos componentes y su comportamiento.

```
        protected void buildView() {
                setSizeFull();

                VerticalSplitPanel verticalSplitPanel
                        = new VerticalSplitPanel();
                verticalSplitPanel.setSizeFull();

                addComponent(verticalSplitPanel);

                buildTableAutores();
                verticalSplitPanel.addComponent(tableAutores);

                buildButtons();
                addComponent(addButton, "top:0;right:70px;");
                addComponent(deleteButton, "top:0;right:5px;");

                buildEditor();

                verticalSplitPanel.addComponent(editorLayout);
        }
```

Básicamente, el componente que estamos programando en AutorEditor es un panel dividido verticalmente en dos. En la parte superior mostraremos una tabla con algunos de los datos de todos los autores, mientras que en la parte inferior mostraremos una vista con un formulario para modificar los datos, una imagen del autor (si se dispone de ella) y un listado con las obras disponibles en la biblioteca (si hay alguna).



Vista del componente AutorEditor con el primer autor seleccionado

También se añaden dos botones en la parte superior derecha para introducir autores nuevos o borrar uno seleccionado.

Para hacer la clase más legible, el método buildView contiene llamadas a funciones internas de la clase que construyen, respectivamente, la tabla de autores, los botones superiores de creación y borrado, y el componente inferior.

No nos detendremos aquí a desgranar estas funciones, puede encontrarse el código correspondiente a las clases AutorEditor y ObraEditor en el anexo E.

Sí que nos detendremos, por ser de sumo interés, en la forma en la que se interactúa con los servicios web.

8.4.1.2.1. Llamada a los servicios web con RESTEasy

La clase AutorEditor realiza todas las operaciones relacionadas con la entidad Autor a través de una clase de servicio, que hemos instanciado en las primeras líneas de código de la clase:

```
private AutorService autorService = new AutorServiceImpl();
```

Y es esta clase la que hemos utilizado para poblar la tabla de autores del componente superior. Lo hemos hecho, concretamente, en el método initContainer.

```
protected void initContainer(){
   autorBeanItemContainer
       = new BeanItemContainer<Autor>(Autor.class, autorService.getAll());
   tableAutores = new Table("Autores", autorBeanItemContainer);

   …
```

La interfaz AutorService declara los siguientes métodos:

```
package es.lta.biblioteca.admin.service;

import java.util.List;
import es.lta.biblioteca.model.Autor;

public interface AutorService {

        public Autor addAutor(Autor autor);
        public Autor getAutor(Integer idAutor);
        public List<Autor> getAll();
        public Autor updateAutor(Autor autor);
        public void deleteAutor(Integer idAutor);
        public void deleteAutor(Autor autor);

}
```

que son implementados por la clase AutorServiceImpl. Esta implementación comienza de la siguiente manera:

```
package es.lta.biblioteca.admin.service.impl;

import java.util.HashSet;
import java.util.List;

import org.jboss.resteasy.client.ProxyFactory;

import es.lta.biblioteca.model.Autor;
import es.lta.biblioteca.model.Obra;
import es.lta.biblioteca.model.Participacion;
import es.lta.biblioteca.admin.Config;
import es.lta.biblioteca.admin.service.AutorService;
import es.lta.biblioteca.service.rest.AutorServiceREST;

public class AutorServiceImpl implements AutorService {

   private AutorServiceREST autorServiceREST =
      ProxyFactory.create(AutorServiceREST.class,
      Config.getBaseURL());

…
```

Aquí estamos utilizando una característica de RESTEasy muy útil para crear clientes a servicios REST. Consiste en utilizar una interfaz anotada con JAX-RS para instanciar un *proxy* que nos abstraiga completamente de las llamadas al servicio web. Recordemos que a la hora de construir los servicios web generamos también unas interfaces, igualmente anotadas, en una librería separada de la implementación de los servicios.

Así pues, indicamos al método *create* de ProxyFactory la interfaz que contiene las anotaciones y los métodos expuestos, y la URL en la que se encuentran los servicios expuestos. Esta URL la provee el método getBaseURL de la clase Config. La clase Config es tan sencilla como lo siguiente:

```
package es.lta.biblioteca.admin;

public class Config {

	private static String baseURL=
		"http://localhost:8080/biblioteca-rest-services";

	public static String getBaseURL() {
		return baseURL;
	}
}
```

De este modo, y como ya hemos dicho antes, nos abstraemos de las llamadas a los servicios web utilizando directamente los métodos definidos en la interfaz AutorServiceREST. Veamos, por ejemplo, la implementación en AutorServiceImpl del método getAll:

```
@Override
public List<Autor> getAll() {
   List<Autor> autores = autorServiceREST.loadAll();
```

```
    for(Autor autor : autores){
       autor.setObras(
        new HashSet<Obra>(autorServiceREST.getObras(autor.getIdAutor())));
       autor.setParticipaciones(
        new HashSet<Participacion>(
           autorServiceREST.getParticipaciones(autor.getIdAutor())));
    }
    return autores;
}
```

RESTEasy es quien se encarga de hacer las llamadas a los servicios web y de realizar el *marshalling* y *unmarshalling* pertinente de las clases del modelo que se manejan. Esta serialización la realiza en función de las anotaciones @Consumes y @Produces que encuentre en la interfaz utilizada para crear el *proxy*. En nuestro caso, como hemos utilizado dos formatos distintos, es de esperar que utilice el primero definido, que es XML.

8.4.1.3. Despliegue en Tomcat

Programada ya la aplicación web, la compilamos y la empaquetamos, junto con sus dependencias, en un fichero WAR. De este proceso se encargará Maven. El WAR resultante lo debemos desplegar en un servidor. Podríamos desplegarlo en el mismo servidor en el que hemos desplegado los servicios web, pero queremos poner de manifiesto la independencia entre un desarrollo y otro, por lo que lo desplegaremos en un servidor distinto.

Ya que la aplicación web no hace uso de característica alguna de Java EE (recordemos que la comunicación con los servicios REST se hace a través de RESTEasy, cuya librería implicada se encuentra en el WAR) nos basta con desplegarlo en un Tomcat.

Utilizaremos la versión 7.0.34 (la última en el momento en el que se desplegó por primera vez la aplicación). Ya que los dos servidores (TomEE, para los servicios REST, y Tomcat) estarán ejecutándose a la vez en la misma máquina, cambiaremos los puertos que utiliza por defecto Tomcat. Para ello debemos editar el fichero $TOMCAT_HOME/conf/server.xml, donde $TOMCAT_HOME es el directorio raíz de Tomcat.

Allí cambiaremos las referencias a los puertos 8005, 8080 y 8009 por 8015, 8081 y 8019 respectivamente.

Por último, copiaremos nuestro WAR (llamado biblioteca-admin.war) con la aplicación web a la carpeta $TOMCAT_HOME/webapps y arrancaremos el servidor con el script $TOMCAT_HOME/bin/startup.bin. Nuestra aplicación se desplegará y estará disponible en http://localhost:8081/biblioteca-admin.

8.4.2. Aplicación Android

Abordamos ahora el último desarrollo de nuestro sistema, la aplicación para Android. Para implementarla haremos uso de las herramientas que proporciona la propia Google[66]. En el momento de comenzar el desarrollo son un SDK[67] y un *plugin* para Eclipse. En este caso no utilizaremos Maven.

Creamos el proyecto con el *wizard* correspondiente que nos proporciona el *plugin* para Eclipse, indicando en el proceso, además, el icono que deseamos para nuestra aplicación. Hemos descargado un paquete de iconos de libre distribución y hemos utilizado uno de ellos. Puede encontrarse este paquete de iconos en http://www.smashingmagazine.com/2009/01/05/stationery-icons-soccer-icons-and-atlantic-wordpress-theme/.

No es el objetivo de esta memoria elaborar un curso de programación Android. En lo que sigue, trataremos el desarrollo realizado en función del interés que suscita como parte del sistema completo que estamos desarrollando.

El elemento principal de una aplicación Android es su fichero AndroidManifest.xml, en el que se indican sus aplicaciones y otros detalles. En nuestro caso es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.lta.biblioteca.android"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:name="es.lta.biblioteca.android.BibliotecaApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="es.lta.biblioteca.android.AutorActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
```

[66] Todas las herramientas y documentación para el desarrollo de aplicaciones para Android pueden encontrarse en http://developer.android.com.
[67] Software Development Kit

```
            </intent-filter>
        </activity>
        <activity
            android:name="es.lta.biblioteca.android.AutorDetalleActivity"
            android:label="@string/title_activity_autor_detalle" >
        </activity>
        <activity
            android:name="es.lta.biblioteca.android.ObraActivity"
            android:label="@string/title_activity_obra" >
        </activity>
        <activity
            android:name="es.lta.biblioteca.android.ObraDetalleActivity"
            android:label="@string/title_activity_obra_detalle" >
        </activity>
    </application>

</manifest>
```

En primer lugar encontramos las versiones para las que nuestra aplicación va a funcionar. Hemos indicado que el nivel mínimo de la SDK es 9, que corresponde a la versión 2.3.2 de Android. El nivel máximo es 17, que corresponde a la versión 4.2.x de Android.

Hemos escogido estas versiones por los terminales que utilizaremos para ejecutar esta aplicación, que son:

- Motorola Droid Pro, con Android 2.3.2
- Nexus 4, con Android 4.2.2[68]

Seguidamente indicamos el único permiso que necesitará nuestra aplicación, que es la salida a internet.

Por último declaramos nuestras actividades, que serán cuatro; dos por entidad. Habrá una destinada a listar los autores y otra para mostrar los detalles de un autor seleccionado de la lista. Otras dos actividades realizarán las funciones análogas para las obras.

La actividad que lista los autores (es.lta.biblioteca.android.AutorActivity) es la actividad que se ejecuta por defecto, y es por ella por la que comenzaremos a revisar el desarrollo hecho.

AutorActivity extiende la actividad ListActivity propia de Android, que lista en pantalla una serie de elementos. Al ejecutar la actividad se ejecuta el método onCreate, cuyo código es el siguiente:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setTitle(string.title_activity_autor);
      setContentView(R.layout.activity_autor);
```

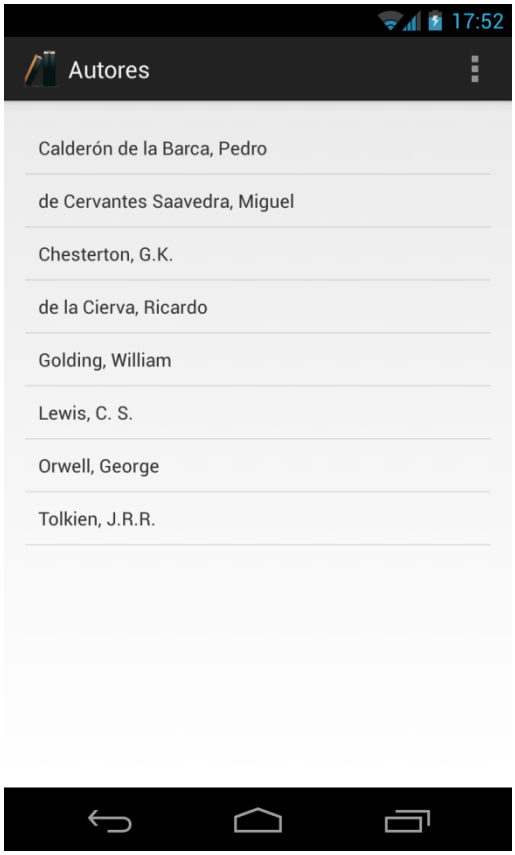
[68] Con el que están tomadas las capturas de pantalla que veremos en adelante.

```
            BibliotecaApplication application = (BibliotecaApplication)
getApplication();
            autores = application.getAutorClient().loadAll();
            Collections.sort(autores, new Autor.ApellidoComparator());
            fillData();
            registerForContextMenu(getListView());
        }
```

Es en este método en el que se obtienen los autores. Para ello vemos que estamos obteniendo un cliente de la clase BibliotecaApplication con el método getAutorClient. Volveremos a esto más adelante. Por el momento, daremos por hecho que el método getAll del cliente nos devuelve la lista de todos los autores. Los almacenamos en una lista local de la clase, los ordenamos por apellido y con el método fillData, que hemos creado, y registerForContextMenu los mandamos a la vista. Esta vista está definida en un fichero xml muy simple:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".AutorActivity" >

    <ListView
        android:id="@+id/android:list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />

    <TextView
        android:id="@+id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no_autores" />

</RelativeLayout>
```

Como vemos, contiene una lista (que hemos rellenado antes con el método fillData) y un texto en caso de que la lista esté vacía.



Vista de la actividad AutorActivity

En caso de que seleccionemos uno de los autores listados se ejecutará la actividad de detalle. Este evento lo controlamos con un *listener* en el que llamaremos a la actividad de detalle pasando como parámetro el identificador del autor.

```
@Override
protected void onListItemClick
            (ListView l, View v, int position, long id) {
      super.onListItemClick(l, v, position, id);
      Intent i = new Intent(this, AutorDetalleActivity.class);
      i.putExtra(
            Constants.SELECTED_AUTOR,
            autores.get(position).getIdAutor());
      startActivity(i);
}
```

Por último, en el menú de opciones hemos puesto un enlace a la actividad que muestra el listado de obras. El inicio de la actividad se controla también con un *listener*:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case R.id.action_obras:
            Intent i = new Intent(this, ObraActivity.class);
            startActivity(i);
            return true;
```

```
			default:
				break;
			}
			return super.onOptionsItemSelected(item);
		}
```

8.4.2.1. Llamada a los servicios web desde Android

Volvamos, sin embargo, al cliente que hemos obtenido para obtener la lista de los autores. Como hemos dicho, lo hemos obtenido de la clase BibliotecaApplication, que extiende la clase Application de Android. Se trata de una clase que se instanciará al ejecutar la aplicación y que utilizaremos para servir los clientes a los servicios REST que tenemos expuestos.

```
package es.lta.biblioteca.android;

import android.app.Application;
import es.lta.biblioteca.android.service.rest.AutorClient;
import es.lta.biblioteca.android.service.rest.ObraClient;
import es.lta.biblioteca.android.service.rest.impl.AutorJSONClient;
import es.lta.biblioteca.android.service.rest.impl.ObraJSONClient;

public class BibliotecaApplication extends Application{

	private AutorClient autorClient;
	private ObraClient obraClient;

	@Override
	public void onCreate() {
		super.onCreate();
		autorClient = new
AutorJSONClient("http://1.0.0.12:8080/biblioteca-rest-services");
		obraClient = new
ObraJSONClient("http://1.0.0.12:8080/biblioteca-rest-services");
	}

	public AutorClient getAutorClient() {
		return autorClient;
	}

	public ObraClient getObraClient() {
		return obraClient;
	}
}
```

Como vemos, hemos creado dos clases nuevas: AutorClient y ObraClient. Se trata de interfaces sencillas. Veamos, a modo de ilustración, la correspondiente a AutorClient:

```
	package es.lta.biblioteca.android.service.rest;

	import java.util.List;

	import es.lta.biblioteca.android.model.Autor;
	import es.lta.biblioteca.android.model.Obra;

	public interface AutorClient {
```

```
        public List<Autor> loadAll();
        public Autor load(Integer id);
        public List<Obra> getObras(Integer id);
}
```

Notemos en este punto que las clases del dominio que manejamos en la aplicación Android son propias de la aplicación. Siguen siendo los mismos POJOs que creamos al principio, pero con algunos métodos que hemos añadido. En el caso del autor, por ejemplo, hemos añadido un método para obtener el nombre del autor en formato 'apellidos, nombre' (getDisplayName), otro para el formato 'nombre apellidos' (getFullName) y una clase interna que implementa la interfaz Comparator para realizar el orden en el listado de autores por apellido (ApellidoComparator). También se incluyen como strings estáticos los nombres de los parámetros de la representación JSON del objeto Autor.

La interfaz AutorClient es implementada por la clase AutorJSONClient, cuyo constructor toma como parámetro la URL en la que se encuentran desplegados los servicios web REST. Veamos la implementación que se hace del método loadAll:

```
@Override
public List<Autor> loadAll() {

    List<Autor> autores = new ArrayList<Autor>();
      try {
            JSONObject root =new JSONObject(
                              doGet(baseURL.concat(serviceURL)));
            JSONArray autoresJson =
                              root.getJSONArray(Autor.AUTOR);
            Autor autor;
            for (int i = 0; i < autoresJson.length(); i++) {
                  autor = parse(autoresJson.getJSONObject(i));
                  if(autor!=null){
                        autores.add(autor);
                  }
            }
      } catch (Exception e) {
            e.printStackTrace();
      }
      return autores;
}
```

Como vemos, en este método tratamos con objetos JSON. En primer lugar, invocamos al método doGet proporcionándole como parámetro un string. Este string lo obtenemos al concatenar la variable baseURL (que hemos inicializado en el constructor de la clase con la URL en la que se encuentran desplegados los servicios) y la variable serviceURL que contiene el string "/autor". El método doGet lo veremos en detalle más adelante. Por ahora, basta con saber que retorna un string en formato JSON con todos los autores, como el siguiente:

```
{
    "autor": [
        {
            "idAutor": 1,
            "nombre": "Miguel",
            "apellido1": "Cervantes, de",
            "apellido2": "Saavedra",
            "imagenUrl":
"http://upload.wikimedia.org/wikipedia/commons/6/66/Cervates_jauregui.jpg"
,
            "typeSettings": "blablabla"
        },
        {
            "idAutor": 4,
            "nombre": "J.R.R.",
            "apellido1": "Tolkien",
            "imagenUrl": "http://www.fusion-freak.com/images/stories/2012-
10/Alaitz/tolkien/John-Ronald-Reuel-Tolkien-10.jpg"
        },
        {
            "idAutor": 5,
            "nombre": "William",
            "apellido1": "Golding",
            "imagenUrl": "http://2.bp.blogspot.com/-vcvSNLjlOuI/UD-
kGrxdEqI/AAAAAAAAEEk/sblIZsyWrJg/s1600/0_Golding-William1.jpg"
        },
        {
            "idAutor": 6,
            "nombre": "George",
            "apellido1": "Orwell",
            "imagenUrl": "http://tanpersonal.com/wp-
content/uploads/2013/03/george-orwell.jpg"
        },
        {
            "idAutor": 7,
            "nombre": "C. S.",
            "apellido1": "Lewis",
            "imagenUrl":
"http://ultimaspaginas.files.wordpress.com/2007/08/lewis_1_nocap.jpg"
        },
        {
            "idAutor": 8,
            "nombre": "Ricardo",
            "apellido1": "Cierva, de la",
            "apellido2": "",
            "imagenUrl":
"http://image.casadellibro.com/img/autores/delacierva.jpg"
        },
        {
            "idAutor": 9,
            "nombre": "G.K.",
            "apellido1": "Chesterton",
            "imagenUrl": "http://4.bp.blogspot.com/-D8lY_ktI_-
8/TzuoIz0zGDI/AAAAAAAAA88/d2h99cWWT4I/s400/Chesterton.jpg"
        },
        {
            "idAutor": 10,
```

```
            "nombre": "Pedro",
            "apellido1": "Calderón",
            "apellido2": "Barca, de la",
            "imagenUrl": "http://www.laaventuradelahistoria.es/wp-
content/uploads/2012/04/JUAN-RANA-CALDER%C3%93N-DE-LA-BARCA.jpg"
        }
    ]
}
```

A partir de ese string creamos un objeto JSONObject, que indicaremos después que contiene un array de autores, identificado con "autor" (string que tenemos definido en Autor.AUTOR). Finalmente, iteramos sobre este array *parseando* sus objetos y construyendo la lista de autores. La función *parse* es la encargada de convertir un objeto JSONObject en un Autor.

```
protected static Autor parse(JSONObject autorJson){
        try {
            Autor autor =
                    new Autor(autorJson.getString(Autor.NOMBRE));
            autor.setIdAutor(autorJson.getInt(Autor.ID_AUTOR));
            autor.setApellido1(autorJson.optString(Autor.APELLIDO_1));
            autor.setApellido2(autorJson.optString(Autor.APELLIDO_2));
            autor.setImagenUrl(autorJson.optString(Autor.IMAGEN_URL));
            autor.setMetaInf(autorJson.optString(Autor.META_INF));
            autor.setTypeSettings(
                autorJson.optString(Autor.TYPE_SETTINGS));
            return autor;
        } catch (JSONException e) {
            return null;
        }
}
```

La diferencia entre los métodos getString y optString de la clase JSONObject radica en que el primero eleva una excepción en caso de que no se encuentre el parámetro deseado en el objeto JSON, mientras que el segundo devuelve un string vacío.

Recuperemos el método doGet que habíamos dejado antes para más adelante. Este método pertenece a la clase abstracta AbstractClient que extienden tanto AutorJSONClient como ObraJSONClient. La clase es la siguiente:

```
package es.lta.biblioteca.android.service.rest;

import …

public abstract class AbstractClient {

    protected String doGet(String url) {
        try {
            return new GetterTask().execute(url).get();
        } catch (Exception e) {
            return "";
        }
    }
```

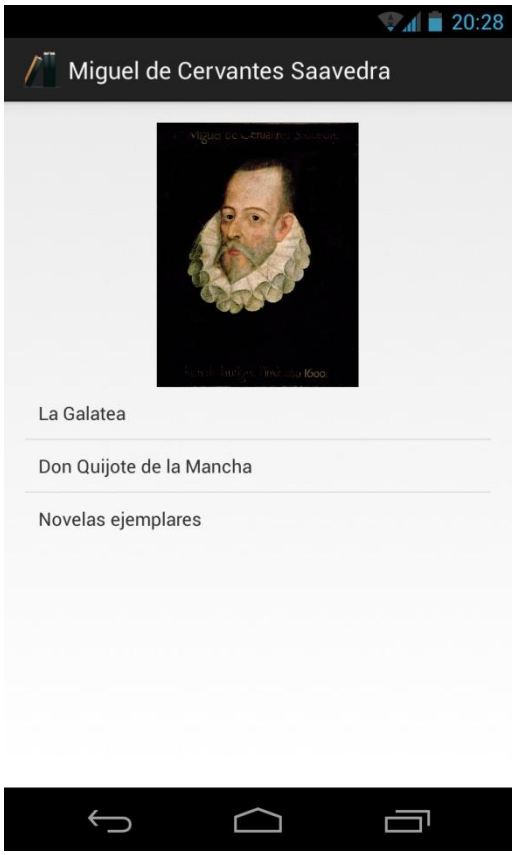
```
	private class GetterTask extends AsyncTask<String, Void, String>{

		@Override
		protected String doInBackground(String... params) {
			String url = params[0];
			StringBuilder builder = new StringBuilder();
			HttpClient client = new DefaultHttpClient();
			HttpGet httpGet = new HttpGet(url);
			httpGet.setHeader("Accept", "application/json");
			try {
				HttpResponse response = client.execute(httpGet);
				StatusLine statusLine =
							response.getStatusLine();
				int statusCode = statusLine.getStatusCode();
				if (statusCode == 200) {
				   HttpEntity entity = response.getEntity();
				   InputStream content = entity.getContent();
				   BufferedReader reader = new BufferedReader(
				               new InputStreamReader(content));
				   String line;
				   while ((line = reader.readLine()) != null) {
				      builder.append(line);
				   }
				} else {
				   Log.e(BibliotecaApplication.class.toString(),
				             "Failed to download file");
				}
				return builder.toString();
			} catch (ClientProtocolException e) {
				e.printStackTrace();
			} catch (IOException e) {
				e.printStackTrace();
			}
			return "";
		}
	}
}
```

El cometido de los métodos de esta clase es el de realizar una request en segundo plano, en un hilo de ejecución distinto al principal de la aplicación. Desde la versión 3.2 de Android (correspondiente a la versión 13 de la SDK) es obligatorio realizar en segundo plano las peticiones a recursos externos a la aplicación. En nuestra aplicación es el caso de las llamadas a los servicios REST y la carga de las imágenes de los autores.

Abreviando el método doInBackground de la clase interna GetterTask, éste realiza una *request* de tipo GET, estableciendo que espera un contenido de tipo JSON. Si la *response* contiene el código de retorno HTTP 200 (correspondiente a 'OK') lee el contenido de la misma y construye un string que devolverá como resultado de la ejecución.

Como hemos dicho, esta restricción de obtener un recurso externo haciendo uso de un hilo separado de ejecución también afecta a la carga de imágenes. Recuperemos la actividad del autor para verlo.

Como dijimos, al seleccionar uno de los autores listados la aplicación nos lleva a una vista con el detalle del autor. En esta vista visualizamos su nombre, una imagen y una lista con las obras que tenemos en la biblioteca.



Vista de la actividad AutorDetalleActivity

Recordemos que la actividad AutorActivity llama a la actividad AutorDetalleActivity pasándole un parámetro con el identificador del autor. AutorDetalleActivity recoge ese parámetro y dibuja la pantalla con la visualización del detalle en el método onResume.

```
@Override
protected void onResume() {
	super.onResume();
	Bundle extras = getIntent().getExtras();
	if(extras!=null){
		Integer autorId =
			(Integer)extras.get(Constants.SELECTED_AUTOR);
		BibliotecaApplication application =
			(BibliotecaApplication) getApplication();
		autor=application.getAutorClient().load(autorId);
		setTitle(autor.getFullName());
		obras=application.getAutorClient().getObras(autorId);

		fotoView = (ImageView)findViewById(R.id.fotoAutor);
		if(autor.getImagenUrl()!=null
			&& !autor.getImagenUrl().equalsIgnoreCase("")){
			try {
```

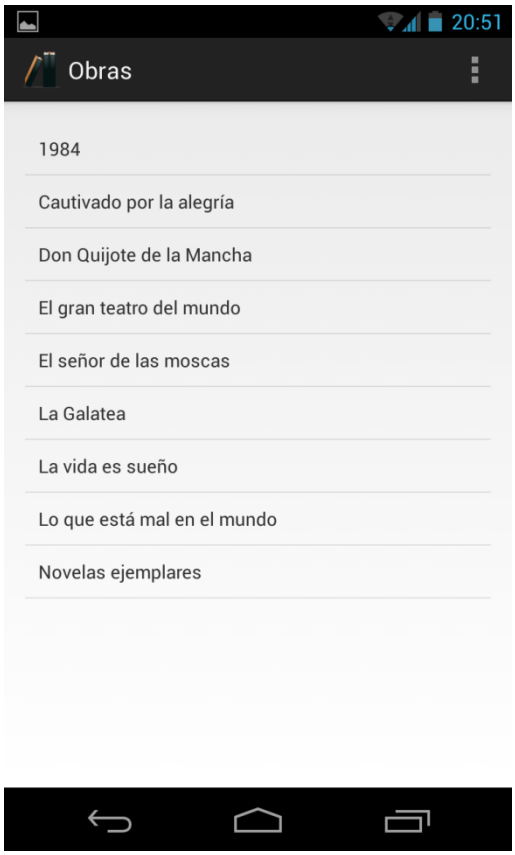
```
                        fotoView.setImageDrawable(
                                Utils.getDrawableFromURL(
                                        autor.getImagenUrl(), autor.getNombre()));
                        } catch (Exception e) {
                        }
                }

                fillData();
        }
}
```

Una vez recuperado el identificador del autor hacemos uso del cliente REST para obtener el autor y sus obras. Con la URL de la imagen del autor obtenemos la imagen. Para ello usamos el método getDrawableFromURL de la clase de utilidades Utils que hemos creado.

```
package es.lta.biblioteca.android.helper;

import java.util.concurrent.ExecutionException;

import android.graphics.drawable.Drawable;
import android.os.AsyncTask;

public class Utils {

        public static Drawable getDrawableFromURL(
                                                String url, String src_name)
                        throws java.net.MalformedURLException,
                                java.io.IOException,
                                InterruptedException,
                                ExecutionException {

                return new Utils().drawableFromURLExecutor(url, src_name);
        }

        private Drawable drawableFromURLExecutor(
                                                String url, String src_name)
                        throws InterruptedException, ExecutionException{
                return new DrawableFromURLTask().execute(url,src_name).get();
        }

        private class DrawableFromURLTask extends
                        AsyncTask<String, Void, Drawable> {

                @Override
                protected Drawable doInBackground(String... params) {
                        try {
                          return Drawable.createFromStream(
                            ((java.io.InputStream) new java.net.URL(
                            params[0]).getContent()),params[1]);
                        } catch (Exception e) {
                                return null;
                        }
                }
        }
}
```

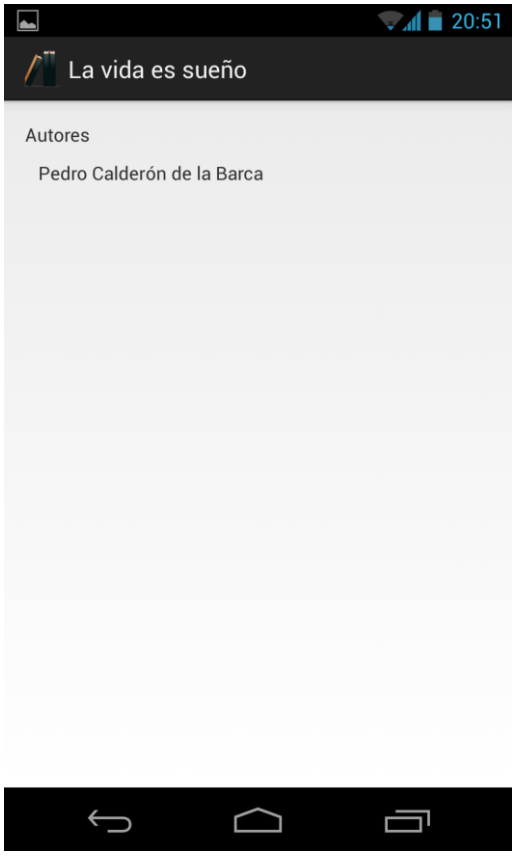
El método getDrawableFromURL instancia la misma clase Utils para que la clase interna DrawableFromURLTask se instancie y esté disponible. Después ejecuta el método doInBackground para que el acceso al recurso externo se haga en otro hilo. En este caso no tenemos que realizar nosotros a mano la *request*, sino que el método createFromStream de la clase Drawable de Android creará un objeto Drawable (que es el que devolveremos) a partir de la URL que le hemos pasado.

Las actividades ObraActivity y ObraDetalleActivity tienen comportamientos análogos a los vistos para el caso del autor, tanto en el *front-end* como en el *back-end*. ObraActivity muestra una lista con las obras disponibles en la biblioteca.



Vista de la actividad ObraActivity

Al seleccionar una de las obras la aplicación pasa a la actividad ObraDetalleActivity, donde se muestra el título de la obra y la lista de sus autores.



Vista de la actividad ObraDetalleActivity

Con esto terminamos la exposición del desarrollo realizado para Android. La compilación de la aplicación resulta en un fichero .apk, que podemos ejecutar en un emulador de Android que hayamos configurado con las herramientas de la SDK o que podemos instalar directamente en un terminal Android tras habilitar en los ajustes la instalación de aplicaciones desde orígenes desconocidos.

9. Pruebas realizadas

A lo largo del desarrollo se han ido realizando pruebas funcionales de todos los elementos implicados en el sistema. Hemos efectuado pruebas de creación y recuperación de autores y obras a través de la llamada directa a los servicios web. Hemos comprobado el comportamiento de los componentes de la aplicación web programada en Vaadin y el resultado, de nuevo, de crear y recuperar autores y obras desde la aplicación. Hemos navegado por la aplicación para Android, consultando los distintos datos del sistema siguiendo distintos patrones de comportamiento.

Fruto de la realización de estas pruebas hemos detectado numerosos problemas, que hemos citado antes en esta memoria. Los cambios en los *CascadeType* y *FetchType* de los mapeos JPA han sido motivados por los resultados de las pruebas. Lo mismo que el marcado de elementos *transient* en JAXB, o el uso de clases *wrapper* para la serialización de listas.

El desarrollo de la aplicación en Vaadin ha consistido, de hecho, en un ejercicio continuo de prueba y error que en ocasiones, todo hay que decirlo, ha resultado desquiciante.

10. Seguimiento del proyecto

Aunque no se ha realizado un marcaje exhaustivo de las horas dedicadas al proyecto, si podemos establecer una aproximación a las horas consumidas por las siguientes tareas:

Entorno de desarrollo (incluye MV)	10 horas
Módulo de datos	100 horas
Módulo de servicios REST	40 horas
Aplicación web	120 horas
Aplicación Android	50 horas
Redacción de la memoria	50 horas

En total suman 370 horas.

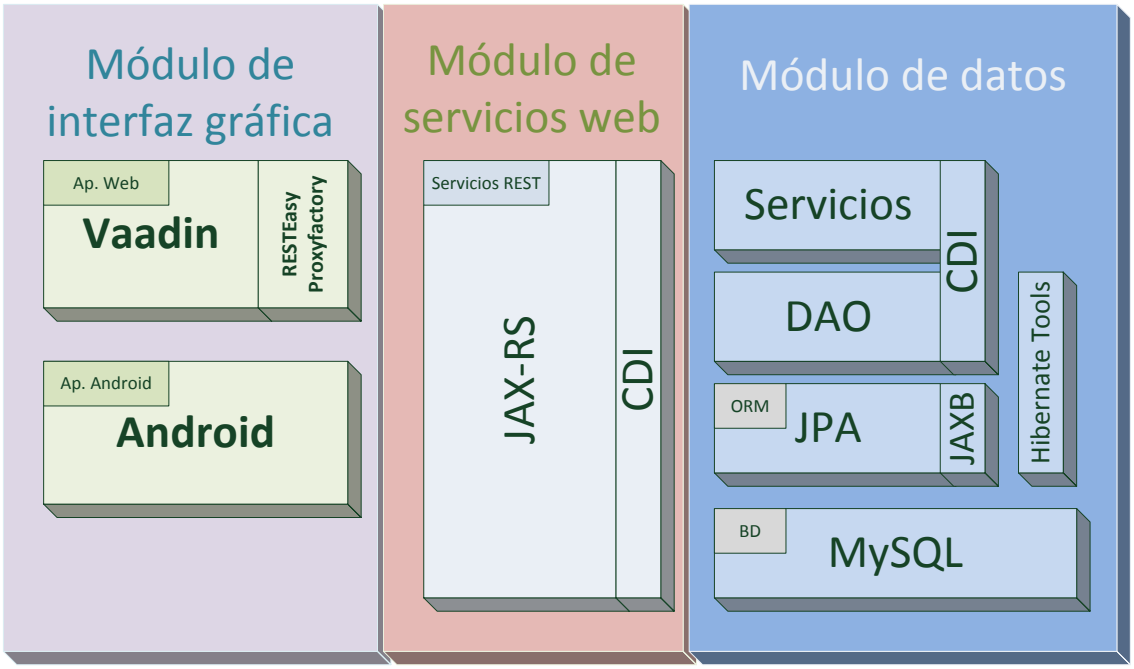
Si lo comparamos con la estimación original de la dedicación semanal que señalamos en el DOP (ver anexo), vemos que la dedicación ha sido menor. En el DOP se indicaba una dedicación de entre 12 y 16 horas semanales. Si tomamos como semanas hábiles para realizar el cálculo las 40 que van de octubre de 2012 a junio de 2013 (ambos inclusive), resultan entre 480 (que vamos a tomar por referencia) y 640 horas. Por tanto, la dedicación ha sido un 23% menor. Las responsabilidades laborales han motivado este descenso en la dedicación.

Así como en la dedicación se ha experimentado una desviación, en las fechas previstas como hitos en el desarrollo también se ha dado una desviación. Al haber sido la dedicación menor, las fechas se han prolongado, y el proyecto, que se estimó finalizar el 31 de mayo de 2013 se ha extendido mes y medio más.

11. Retrospectiva y conclusiones

Con la exposición del desarrollo de la aplicación Android hemos terminado de abordar todos los pasos de la implementación que abarca este PFC. Hemos tratado todas sus capas, desde los datos hasta la presentación. Hemos tratado, como proponíamos en el objetivo, las decisiones tomadas para definir la arquitectura e implementarla. Hemos hablado también de las alternativas, cuando las ha habido, y hemos explicado las situaciones problemáticas que nos hemos encontrado por el camino.

A lo largo del desarrollo hemos adoptado un *stack* tecnológico para nuestro sistema, que ha dado forma a aquel boceto que describíamos en las primeras páginas de esta memoria. Si trasladamos esas tecnologías al diagrama de módulos que esbozábamos en el apartado de arquitectura podemos hacernos una idea más gráfica del papel que juega cada una de estas tecnologías en el sistema.



Este *stack* constituye la decisión que hemos tomado a la hora de enfrentarnos a la implementación del sistema. Una decisión plenamente válida, aunque mejorable a tenor de los problemas que hemos ido encontrando.

Los principales problemas han venido justamente de la utilización de las APIs de Java EE, criterio que habíamos establecido para la elección de soluciones a adoptar en la implementación. Son problemas, sin embargo, que las implementaciones de dichas APIs atajan extendiendo la funcionalidad del estándar que define Java EE.

Así, por ejemplo, si hubiéramos adoptado Hibernate en lugar de JPA como solución ORM para nuestro sistema, nos habríamos ahorrado el lidiar con la coherencia de las relaciones entre instancias. Además de que Hibernate maneja correctamente los *lazy fettching*.

Otro ejemplo lo constituye JAXB y su aversión a las relaciones entre objetos serializables, aspecto que su implementación MOXy ataja.

Por tanto, de tener que formular de nuevo el criterio para escoger soluciones para nuestro desarrollo, éste pasaría a ser el de primar soluciones maduras y contrastadas por encima de otras que lo estén en menor medida o de la correspondiente API de Java que implementase la solución si ésta API ofrece menor rendimiento y comodidad a la hora de programar.

Anexo A: continuidad del proyecto

Como hemos dicho en el primer apartado de esta memoria, el dedicado a su objetivo, este proyecto nace de una necesidad real. Por tanto, el proyecto que hemos comenzado con este PFC tendrá continuidad, hasta que se cubran las necesidades originales de gestión de la biblioteca.

Así pues, el siguiente paso lógico a dar es el de implementar las funciones análogas que en este PFC hemos implementado para Autor y Obra para el resto de entidades. Expandiremos así los servicios web expuestos y las pantallas de administración y consulta de las aplicaciones web y Android.

Sin embargo, hay un margen mayor en el cual este sistema puede crecer y/o mejorar, y lo hemos identificado en varios aspectos que detallamos a continuación.

A.1. MariaDB

MySQL no ha experimentado, desde su integración en Oracle, el ritmo de evolución que se le suponía como uno de los SGBD referentes open source. Es por ello que, de la mano de sus desarrolladores originales, ha surgido MariaDB[69].

MariaDB es un SGBD construido a partir de MySQL, plenamente compatible con él y que está experimentando un mayor soporte y desarrollo, además de tener una sola versión GPL (en lugar de una GPL, con la versión Community, y otra privativa de MySQL).

Dicho esto, contemplamos sustituir la base de datos MySQL por MariaDB.

A.2. Piwigo

En nuestro sistema almacenamos URLs con imágenes para las ediciones y los autores. Se trata de enlaces externos, con lo que nos exponemos a varios riesgos. Uno de ellos es que la URL deje de estar disponible. Otro, que la latencia al recuperar una imagen desde su URL sea alta. No entraremos en temas de licencia de las imágenes porque el sistema es de uso privado.

Para solventar esos dos posibles problemas vemos conveniente disponer de un repositorio controlado de imágenes. Esto podríamos conseguirlo instalando Piwigo[70] en, por ejemplo, la máquina virtual.

Piwigo tiene mayores funcionalidades que las de ser un simple repositorio de imágenes, pero para nuestro cometido es más que suficiente. En primer lugar deberemos subir a este repositorio las imágenes que deseamos tener disponibles. Después obtendremos las URLs

[69] https://mariadb.org
[70] http://es.piwigo.org/

de las mismas. Además, Piwigo puede proveer diferentes enlaces para diferentes tamaños de una imagen con lo que podemos ahorrar en tráfico de datos. Las redimensiones de las imágenes las realiza automáticamente Piwigo.

Con esto evitamos que las imágenes queden súbitamente no disponibles y homogenizamos la latencia en la carga de imágenes.

A.3. Calibre

Para las ediciones que son digitales indicamos una URL con el enlace a su archivo en un formato apropiado para su lectura en un ebook reader. Para centralizar el almacenamiento de estos ficheros optaremos por el software Calibre. Además, este software dispone de un modo servidor, que proporciona una URL única para los libros en formato ebook que contiene.

Así pues, instalaremos Calibre también en la máquina virtual y las URLs que guardaremos para las ediciones digitales corresponderán a las proporcionadas por él.

A.4. Búsqueda

Para agilizar las búsquedas dentro de la aplicación estudiaremos el modo de incluir una funcionalidad de búsqueda indexada. Una posible solución sería la integración de Lucene[71] en el sistema para la indexación de los campos por los que queramos buscar.

A.5. Disponibilidad en internet

Como ya se habrá podido intuir, este sistema es operativo sólo en el caso en el que tanto los servidores como las máquinas que actúan como clientes estén en la misma red. Para hacer el sistema accesible desde otras ubicaciones (especialmente a través de la aplicación Android) habrá que estudiar el modo de publicar los servicios web y la aplicación web: servidores físicos o virtuales necesarios, mapeo de la IP dinámica, enrutamiento de puertos, *firewalls*, etc.

A.6. Autenticación

En el caso de publicar los servicios habrá que estudiar un método de autenticación en el sistema que no resulte tedioso para el usuario. Pueden utilizarse métodos que van desde la autenticación controlada por un servidor frontal web (por ejemplo, Apache[72] que incluya restricciones para el acceso a determinadas rutas) a la integración de soluciones de autenticación.

---

[71] http://lucene.apache.org/core/, de Apache.
[72] Más formalmente Apache HTTPD Server, http://httpd.apache.org/

Anexo B: Maven

A lo largo de esta memoria nos hemos referido en multitud de ocasiones a Maven diciendo, en la mayoría de las ocasiones, que sería el encargado de empaquetar y gestionar las dependencias de nuestros desarrollos. A continuación haremos un breve resumen de las funcionalidades de Maven.

Maven es una herramienta de gestión de proyectos que maneja las dependencias del proyecto, lo compila, empaqueta, ejecuta tests, etc. También puede ejecutar otra serie de tareas con la incorporación de plugins. Todo ello se controla desde un fichero central, pom.xml, que se ubica en la raíz del proyecto.

Se puede descargar desde http://maven.apache.org/. Debemos incluir su directorio de instalación en el *classpath* del sistema. Desde línea de comandos, podemos ejecutar Maven con el comando mvn.

B.1. POM.xml

Como hemos dicho, se trata del archivo central. Aquí indicamos los datos de nuestro proyecto, sus dependencias y las tareas que ha de ejecutar Maven.

Las dependencias de un proyecto se configuran indicando en este archivo los datos del artefacto del que se depende. Maven trata los proyectos como artefactos. Todo proyecto ha de tener un **groupId**, un **artifactId** y una **version** que lo identifique. Veamos un pom.xml sencillo de nuestro proyecto:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>es.lta.biblioteca</groupId>
    <artifactId>model-dao</artifactId>
    <version>1.5</version>
  </parent>
  <artifactId>biblioteca-services</artifactId>
  <name>Biblioteca Services</name>
      <description>
            Servicios para el modelo de dominio de la biblioteca
      </description>
      <inceptionYear>2013</inceptionYear>
      <developers>
            <developer>
                  <id>luisto</id>
                  <name>Luis Tomás García Sánchez</name>
                  <roles>
                        <role>architect</role>
                        <role>developer</role>
                  </roles>
```

```
		</developer>
	</developers>

	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<configuration>
					<source>1.7</source>
					<target>1.7</target>
				</configuration>
			</plugin>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-jar-plugin</artifactId>
				<version>2.4</version>
				<executions>
					<execution>
						<phase>package</phase>
						<goals>
							<goal>jar</goal>
						</goals>
					</execution>
				</executions>
				<configuration>
					<finalName>${artifactId}</finalName>
				</configuration>
			</plugin>
		</plugins>
	</build>
	<dependencies>
		<dependency>
			<groupId>es.lta.biblioteca</groupId>
			<artifactId>biblioteca-model</artifactId>
			<version>${project.version}</version>
			<scope></scope>
		</dependency>
		<dependency>
			<groupId>es.lta.biblioteca</groupId>
			<artifactId>biblioteca-dao</artifactId>
			<version>${project.version}</version>
			<scope></scope>
		</dependency>
	</dependencies>
</project>
```

En primer lugar indicamos que este pom.xml tiene un pom.xml padre del que hereda más propiedades. Lo veremos luego. El artifactId del artefacto que constituye nuestro proyecto es biblioteca-rest-services-api. Después encontramos varias propiedades más del proyecto, que no es obligatorio indicar, pero que resultan informativas.

En plugins, dentro de la etiqueta build, encontramos la configuración para los *plugins* encargados de la compilación y empaquetado del proyecto. Al encargado de la compilación

le indicamos para qué versión de Java está escrito el código y la versión para la que nos tiene que generar el *bytecode*.

El *plugin* para el empaquetado recibe como configuración tan sólo el formato de nombre que ha de recibir el JAR resultante, que será biblioteca-rest-services-api.jar. Como vemos, los plugins se identifican también como artefactos.

Por último indicamos las dependencias de este proyecto (además de las que hereda del padre, como veremos después). Para cada una de las dependencias indicamos sus parámetros identificadores. Así, las dependencias de este proyecto son los paquetes con el modelo de dominio y los DAOs.

Al ejecutar en la raíz de este proyecto

mvn clean compile package

Maven ejecutará primero la tarea clean, que consiste en borrar todos aquellos archivos que hayan sido generados, bien por el IDE, bien por ejecuciones anteriores de Maven. La tarea compile compilará todo el código. Finalmente, la tarea package empaquetará nuestro proyecto compilado. En este caso, lo empaquetará en un JAR.

Si además de esas tres tareas hubiéramos indicado la tarea install, nuestro paquete se hubiera instalado como artefacto en el repositorio local de artefactos que maneja Maven. Por defecto, todas las dependencias se obtienen de repositorios centrales de Maven disponibles en Internet. La primera vez que se obtiene un artefacto de un repositorio externo se instala en el repositorio local para tener acceso a él más rápidamente en el futuro.

Echemos ahora un vistazo al pom.xml padre.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>es.lta.biblioteca</groupId>
      <artifactId>model-dao</artifactId>
      <version>1.5</version>
      <name>Biblioteca Model and DAO</name>
      <packaging>pom</packaging>
      <description>Modelo de dominio para la biblioteca y
DAOs</description>
      <inceptionYear>2012</inceptionYear>
      <developers>
            <developer>
                  <id>luisto</id>
                  <name>Luis Tomás García Sánchez</name>
                  <roles>
                        <role>architect</role>
                        <role>developer</role>
                  </roles>
```

```
		</developer>
	</developers>
	<modules>
		<module>biblioteca-model</module>
		<module>biblioteca-dao</module>
		<module>biblioteca-services</module>
	</modules>
	<dependencies>
		<dependency>
			<groupId>javax</groupId>
			<artifactId>javaee-web-api</artifactId>
			<version>6.0</version>
			<scope>provided</scope>
		</dependency>
	</dependencies>
</project>
```

Vemos de nuevo que se definen las propiedades de groupId, artifactId y version, así como otra información para el artefacto. Después indicamos los módulos que son hijos de este pom. Se trata de artefactos que comparten el mismo groupId. Indicar los hijos como modulos permite que al ejecutar las tareas para el padre, se ejecuten primero las mismas tareas para los hijos en el orden en el que estos están dispuestos.

Por último, se definen las dependencias que son generales a todos los hijos, permitiendo que los pom hijos no tengan que definirlas. La dependencia común aquí es la versión 6 de la API del perfil web de Java EE.

B.2. Comandos para el proyecto

No entraremos en más detalle acerca de las posibilidades de Maven. Lo expuesto aquí sirve ya de introducción muy básica a Maven[73]. Indicaremos a continuación los comandos a ejecutar para la construcción de los proyectos de este PFC (se indican los comandos para los pom padres):

- model-dao
  Ejecución de las hibernate tools, compilación, empaquetado e instalación de los artefactos (JARs)

  mvn clean package install –Phbtools

- rest-services
  Compilación, empaquetado e instalación de los artefactos (JARs)

  mvn clean compile package install

[73] Puede encontrarse una introducción más extensa en http://www.genbetadev.com/java-j2ee/introduccion-a-maven y http://www.genbetadev.com/java-j2ee/introduccion-a-maven-ii-project-object-model. También se encuentra disponible una completa guía de referencia, bajo licencia CC, con el título Better Builds with Maven. Consultar el apartado de bibliografía.

- biblioteca-admin
  Compilación y empaquetado del artefacto (WAR)

  mvn clean compile package

Anexo C: puesta a punto de la máquina virtual

Como hemos dicho antes, utilizaremos una máquina virtual en la que instalar algunos de los elementos implicados en el proyecto. Concretamente SVN y la base de datos.

La puesta a punto de la máquina se ha llevado a cabo como sigue.

C.1. Virtual Box

Dentro de Virtual Box configuraremos una máquina que albergue el sistema operativo de nuestra máquina virtual. Para ello, mediante la opción nueva de Virtual Box, le damos un nombre (vmPFC) y le indicamos que albergará un sistema Linux, concretamente un Ubuntu de 64 bits. Le daremos la siguiente configuración inicial:

1 CPU con límite de ejecución del 100%
RAM: 2GB (se podrá ampliar posteriormente)
Disco virtual: reservado dinámicamente, de 10GB
Interfaz de red: bridge (puente)

C.2. Ubuntu Server

Descargamos la ISO del S.O., un Ubuntu Server 12.04 y configuramos la máquina virtual para que arranque desde esa ISO.

Comenzará la instalación, en la cual indicaremos que el nombre de la máquina será 'vmPfc'. El usuario por defecto será 'pfc' y su contraseña 'pfcPass'. Durante la instalación indicaremos que también queremos instalar OpenSSH Server y LAMP Server.

De LAMP Server utilizaremos en principio el servidor web Apache y MySQL. PHP puede sernos de utilidad en el futuro si instalamos, por ejemplo, Piwigo.

Cuando nos pida la contraseña para el usuario 'root' de MySQL indicamos 'pfcPass'.

C.3. Samba

Para que nuestra máquina Ubuntu sea accesible desde un terminal Windows a través de su nombre de host necesitaremos instalar Samba para que Ubuntu se adapte al protocolo NetBios que utiliza Windows. Para ello ejecutamos la siguiente orden en la terminal:

```
sudo apt-get install samba
```

Con esto se instalará Samba y se iniciará el servicio. Ya podemos acceder desde Windows a la máquina a través de su nombre (vmPfc).

C.4. JDK

Dejaremos instalada la JDK de Oracle por si más adelante movemos los servidores Tomcat y TomEE a esta máquina.

Por cambios en las licencias, la JDK de Oracle no está en los repositorios de Ubuntu. Hay que añadir repositorios externos.

```
#Habilitar add-apt-repository
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
```

Con esto ya podemos instalar la última versión de la JDK para Java 7.

```
sudo apt-get install oracle-jdk7-installer
```

C.5. MySQL

Configuramos MySQL para que acepte conexiones desde fuera de localhost. Para ello creamos el fichero /etc/mysql/conf.d/custom.cnf (lo tendremos que hacer con el usuario 'root') con el siguiente único contenido:

```
[mysqld]
bind-address=0.0.0.0
```

Ahora hay que dar permisos al usuario 'root' de MySQL para acceder desde otra IP. Ejecutamos

```
mysql -u root –p
```

Y a continuación, cuando nos lo pida, introduciremos la contraseña. Ejecutamos la sentencia:

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%'
   -> IDENTIFIED BY 'pfcPass' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

C.6. SVN

Desde la terminal ejecutamos la sentencia

```
sudo apt-get install subversion subversion-tools libapache2-svn
```

Creamos las carpetas necesarias para almacenar nuestro respositorio en la siguiente ruta: /opt/svn/biblioteca. A continuación, indicamos a SVN que cree un nuevo repositorio:

```
sudo svnadmin create /opt/svn/biblioteca
```

Y configuramos también la ruta para el *trunk*, *branches* y *tags* mediante *commits*:

```
sudo svn mkdir file:///opt/svn/biblioteca/trunk -m "Trunk"
```

```
sudo svn mkdir file:///opt/svn/biblioteca/tags -m "Tags"
sudo svn mkdir file:///opt/svn/biblioteca/branches -m "Branches"
```

Para gestionar mejor los permisos creamos el grupo ‘subversion’:

```
sudo groupadd subversion
```

Y le añadimos un usuario para los *commits*, en este caso ‘luisto’:

```
sudo useradd luisto
sudo usermod -a -Gsubversion luisto
```

Y a continuación lo añadimos a los ficheros de acceso del módulo de SVN para apache, indicando el password ‘luisto’ cuando nos lo pida:

```
sudo htpasswd -c /etc/apache2/dav_svn.passwd luisto
```

Cambiamos el propietario de todo el directorio /opt/svn, de manera recursiva, a www-data, que es el usuario de apache (ya que haremos los *commits* a través de apache) y al grupo ‘subversion’:

```
sudo chown -R www-data:subversion /opt/svn
```

Y cambiamos los permisos para el mismo directorio:

```
sudo chmod 775 -R /opt/svn
```

Para habilitar la modificación posterior de los *commits* (como, por ejemplo, la rectificación de comentarios) necesitamos registrar un *hook*. Al crear el repositorio se crean con él algunas plantillas que nos servirán. En la ruta /opt/svn/biblioteca/hooks copiamos la plantilla pre-revprop-change.tmpl al archivo pre-revprop-change:

```
sudo cp pre-revprop-change.tmpl pre-revprop-change
```

Le damos los permisos adecuados:

```
sudo chmod 775 pre-revprop-change
```

Y le cambiamos los propietarios:

```
sudo chown www-data:subversion pre-revprop-change
```

Por último configuramos el acceso a través de apache al repositorio. Añadiremos al fichero de configuración /etc/apache2/mods-available/dav_svn.conf lo siguiente:

```
##
# Biblioteca
#
<Location /biblioteca>
DAV svn
SVNPath /opt/svn/biblioteca
AuthType Basic
AuthName "Repositorio Subversion del PFC / Biblioteca"
AuthUserFile /etc/apache2/dav_svn.passwd
Require valid-user
</Location>
```

Y reiniciamos apache con

```
sudo service apache2 restart.
```

Una vez reiniciado, podemos acceder al repositorio desde http://vmPfc/biblioteca.

C.7. Snapshot

Una vez realizadas todas estas instalaciones y configuraciones realizaremos un snapshot de la máquina virtual a modo de copia de seguridad. Llamamos al snapshot “vmPFC-20120930”.

Anexo D: Script de creación de base de datos

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

DROP SCHEMA IF EXISTS `biblioteca` ;
CREATE SCHEMA IF NOT EXISTS `biblioteca` DEFAULT CHARACTER SET utf8
COLLATE utf8_spanish_ci ;
USE `biblioteca` ;

-- -----------------------------------------------------
-- Table `biblioteca`.`Obra`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Obra` (
  `idObra` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador del libro.'
,
  `titulo` VARCHAR(150) NOT NULL COMMENT 'Título del libro.' ,
  `metaInf` TEXT NULL COMMENT 'Información adicional.' ,
  `typeSettings` TEXT NULL COMMENT 'Type settings.' ,
  PRIMARY KEY (`idObra`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Autor`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Autor` (
  `idAutor` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador del autor.'
,
  `nombre` VARCHAR(25) NOT NULL COMMENT 'Nombre del autor.' ,
  `apellido1` VARCHAR(25) NULL COMMENT 'Primer apellido del autor.' ,
  `apellido2` VARCHAR(25) NULL COMMENT 'Segundo apellido del autor.' ,
  `imagen_url` VARCHAR(150) NULL ,
  `metaInf` TEXT NULL ,
  `typeSettings` TEXT NULL COMMENT 'Type settings.' ,
  PRIMARY KEY (`idAutor`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Editorial`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Editorial` (
  `idEditorial` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador de la
editorial.' ,
  `nombre` VARCHAR(45) NOT NULL COMMENT 'Nombre de la editorial.' ,
  `ciudad` VARCHAR(45) NULL COMMENT 'Ciudad de origen de la editorial.' ,
  `pais` VARCHAR(25) NULL COMMENT 'País de origen de la editorial.' ,
  `metaInf` TEXT NULL ,
  `typeSettings` TEXT NULL COMMENT 'Type settings.' ,
  PRIMARY KEY (`idEditorial`) )
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `biblioteca`.`Edicion`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Edicion` (
  `idEdicion` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador de la
edición.' ,
  `idEditorial` INT NOT NULL COMMENT 'Identificador de la editorial.' ,
  `titulo` VARCHAR(150) NOT NULL ,
  `subtitulo` VARCHAR(150) NULL ,
  `numero` INT NULL COMMENT 'Número de edición.' ,
  `anno` INT NULL COMMENT 'Año de la edición.' ,
  `idioma` VARCHAR(45) NULL ,
  `isbn` VARCHAR(13) NULL COMMENT 'Código ISBN de la edición.' ,
  `imagen_url` VARCHAR(150) NULL ,
  `ebook_url` VARCHAR(150) NULL ,
  `metaInf` TEXT NULL ,
  `typeSettings` TEXT NULL COMMENT 'Type settings.' ,
  INDEX `fk_Edicion.Editorial` (`idEditorial` ASC) ,
  PRIMARY KEY (`idEdicion`) ,
  CONSTRAINT `fk_Edicion.Editorial`
    FOREIGN KEY (`idEditorial` )
    REFERENCES `biblioteca`.`Editorial` (`idEditorial` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Categoria`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Categoria` (
  `idCategoria` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador de la
categoría.' ,
  `nombre` VARCHAR(45) NULL COMMENT 'Nombre de la categoría.' ,
  `typeSettings` TEXT NULL COMMENT 'Type settings.' ,
  PRIMARY KEY (`idCategoria`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Etiqueta`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Etiqueta` (
  `idEtiqueta` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador de la
etiqueta.' ,
  `nombre` VARCHAR(45) NOT NULL COMMENT 'Nombre de la etiqueta.' ,
  `idCategoria` INT NULL COMMENT 'Categoría a la que pertenece la
etiqueta.' ,
  `idPadre` INT NULL COMMENT 'Identificador de la etiqueta padre.' ,
  `typeSettings` TEXT NULL COMMENT 'Type settings.' ,
  INDEX `fk_Etiqueta.Categoria` (`idCategoria` ASC) ,
  PRIMARY KEY (`idEtiqueta`) ,
  INDEX `fk_Etiqueta.Etiqueta` (`idPadre` ASC) ,
  CONSTRAINT `fk_Etiqueta.Categoria`
    FOREIGN KEY (`idCategoria` )
    REFERENCES `biblioteca`.`Categoria` (`idCategoria` )
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Etiqueta.Etiqueta`
    FOREIGN KEY (`idPadre` )
    REFERENCES `biblioteca`.`Etiqueta` (`idEtiqueta` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Etiquetado`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Etiquetado` (
  `idObra` INT NOT NULL COMMENT 'Identificador del libro etiquetado.' ,
  `idEtiqueta` INT NOT NULL COMMENT 'Identificador de la etiqueta con la
que se etiqueta.' ,
  PRIMARY KEY (`idObra`, `idEtiqueta`) ,
  INDEX `fk_Etiquetado.Etiqueta` (`idEtiqueta` ASC) ,
  INDEX `fk_Etiquetado.Obra` (`idObra` ASC) ,
  CONSTRAINT `fk_Etiquetado.Obra`
    FOREIGN KEY (`idObra` )
    REFERENCES `biblioteca`.`Obra` (`idObra` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Etiquetado.Etiqueta`
    FOREIGN KEY (`idEtiqueta` )
    REFERENCES `biblioteca`.`Etiqueta` (`idEtiqueta` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`SysConfig`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`SysConfig` (
  `idConfig` INT NOT NULL AUTO_INCREMENT COMMENT 'Identificador de la
propiedad.' ,
  `property` VARCHAR(45) NOT NULL COMMENT 'Nombre de la propiedad.' ,
  `value` TEXT NULL COMMENT 'Valor de la propiedad.' ,
  PRIMARY KEY (`idConfig`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Autor_Obra`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Autor_Obra` (
  `idAutor` INT NOT NULL ,
  `idObra` INT NOT NULL ,
  PRIMARY KEY (`idAutor`, `idObra`) ,
  INDEX `fk_Autor_Obra.Obra` (`idObra` ASC) ,
  INDEX `fk_Autor_Obra.Autor` (`idAutor` ASC) ,
  CONSTRAINT `fk_Autor_Obra.Autor`
    FOREIGN KEY (`idAutor` )
    REFERENCES `biblioteca`.`Autor` (`idAutor` )
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Autor_Obra.Obra`
    FOREIGN KEY (`idObra` )
    REFERENCES `biblioteca`.`Obra` (`idObra` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Relacion`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Relacion` (
  `idRelacion` INT NOT NULL AUTO_INCREMENT ,
  `relacion` VARCHAR(45) NOT NULL ,
  `typeSettings` TEXT NULL ,
  PRIMARY KEY (`idRelacion`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Participacion`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Participacion` (
  `idParticipacion` INT NOT NULL AUTO_INCREMENT ,
  `idAutor` INT NOT NULL ,
  `idEdicion` INT NOT NULL ,
  `idRelacion` INT NOT NULL ,
  `typeSettings` TEXT NULL ,

  PRIMARY KEY (`idParticipacion`) ,
  INDEX `fk_Participacion.Edicion` (`idEdicion` ASC) ,
  INDEX `fk_Participacion.Autor` (`idAutor` ASC) ,
  INDEX `fk_Participacion.Relacion` (`idRelacion` ASC) ,
  CONSTRAINT `fk_Participacion.Autor`
    FOREIGN KEY (`idAutor` )
    REFERENCES `biblioteca`.`Autor` (`idAutor` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Participacion.Edicion`
    FOREIGN KEY (`idEdicion` )
    REFERENCES `biblioteca`.`Edicion` (`idEdicion` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Participacion.Relacion`
    FOREIGN KEY (`idRelacion` )
    REFERENCES `biblioteca`.`Relacion` (`idRelacion` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `biblioteca`.`Obra_Edicion`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `biblioteca`.`Obra_Edicion` (
```

```
  `idObra` INT NOT NULL ,
  `idEdicion` INT NOT NULL ,
  PRIMARY KEY (`idObra`, `idEdicion`) ,
  INDEX `fk_Obra_Edicion.Edicion` (`idEdicion` ASC) ,
  INDEX `fk_Obra_Edicion.Obra` (`idObra` ASC) ,
  CONSTRAINT `fk_Obra_Edicion.Obra`
    FOREIGN KEY (`idObra` )
    REFERENCES `biblioteca`.`Obra` (`idObra` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Obra_Edicion.Edicion`
    FOREIGN KEY (`idEdicion` )
    REFERENCES `biblioteca`.`Edicion` (`idEdicion` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

USE `biblioteca` ;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Anexo E: Componentes Vaadin para las entidades Autor y Obra

E.1. AutorEditor.java

```
package es.lta.biblioteca.admin.components.impl;

import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Map.Entry;

import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.data.fieldgroup.BeanFieldGroup;
import com.vaadin.data.fieldgroup.FieldGroup;
import com.vaadin.data.fieldgroup.FieldGroup.CommitEvent;
import com.vaadin.data.fieldgroup.FieldGroup.CommitException;
import com.vaadin.data.util.BeanItemContainer;
import com.vaadin.server.ExternalResource;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.FormLayout;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Image;
import com.vaadin.ui.Notification;
import com.vaadin.ui.Table;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalSplitPanel;
import com.vaadin.ui.themes.Reindeer;

import es.lta.biblioteca.admin.components.EditorInterface;
import es.lta.biblioteca.admin.service.AutorService;
import es.lta.biblioteca.admin.service.impl.AutorServiceImpl;
import es.lta.biblioteca.model.Autor;
import es.lta.biblioteca.model.Obra;

public class AutorEditor extends AbsoluteLayout implements
EditorInterface{

	/**
	 * serialVersionUID autogenerado
	 */
	private static final long serialVersionUID = -8403976236441418033L;

	private AutorService autorService = new AutorServiceImpl();

	private BeanItemContainer<Autor> autorBeanItemContainer;
	private Table tableAutores;
	private Button addButton;
```

```
	private Button deleteButton;

	private Button commit;
	private Button discard;

	private FormLayout form = new FormLayout();
	private BeanFieldGroup<Autor> formFieldGroup = new
BeanFieldGroup<Autor>(Autor.class);

	private String[] tableAutoresFields = new String[] { "idAutor",
"nombre", "apellido1", "apellido2" };
	private String[] tableAutoresHeaders = new String[] {"id",
"Nombre", "Primer apellido", "Segundo apellido"};

	private Map<String,String> formFields = new LinkedHashMap<String,
String>();

	private boolean flagNuevo = false;
	private Autor newAutor;

	private HorizontalLayout editorLayout;
	private Image fotoAutor = new Image();
	private Table tableObras;
	private BeanItemContainer<Obra> obraBeanItemContainer;

	private String[] tableObrasFields = new String[] {"titulo"};
	private String[] tableObrasHeaders = new String[] {"Obras"};

	public AutorEditor() {
		initFormFields();
		setSizeFull();
		initContainer();
		buildView();
	}

	@Override
	public void setSizeFull() {
		 setWidth(100, Unit.PERCENTAGE);
	    setHeight(100, Unit.PERCENTAGE);
	}

	protected void initContainer(){
		autorBeanItemContainer = new
BeanItemContainer<Autor>(Autor.class, autorService.getAll());
		tableAutores = new Table("Autores", autorBeanItemContainer);
		obraBeanItemContainer = new
BeanItemContainer<Obra>(Obra.class, new HashSet<Obra>());
		tableObras = new Table(null, obraBeanItemContainer);

		//Establecer las columnas visibles en orden
		tableAutores.setVisibleColumns(tableAutoresFields);
		tableAutores.setColumnHeaders(tableAutoresHeaders);
		tableObras.setVisibleColumns(tableObrasFields);
		tableObras.setColumnHeaders(tableObrasHeaders);
	}

	public void reView(){
```

```
            editorLayout.setVisible(false);
            tableAutores.setValue(null);

            autorBeanItemContainer.removeAllItems();
            autorBeanItemContainer.addAll(autorService.getAll());
      }

      protected void buildView() {
            setSizeFull();

            VerticalSplitPanel verticalSplitPanel = new
VerticalSplitPanel();
            verticalSplitPanel.setSizeFull();

            addComponent(verticalSplitPanel);

            buildTableAutores();
            verticalSplitPanel.addComponent(tableAutores);

            buildButtons();
            addComponent(addButton, "top:0;right:70px;");
            addComponent(deleteButton, "top:0;right:5px;");

            buildEditor();

            verticalSplitPanel.addComponent(editorLayout);
      }

      private void initFormFields(){
            formFields.put("nombre", "Nombre");
            formFields.put("apellido1", "Primer apellido");
            formFields.put("apellido2", "Segundo apellido");
            formFields.put("imagenUrl", "URL imagen");
            formFields.put("metaInf", "Meta inf.");
      }

      @SuppressWarnings("serial")
      private void buildTableAutores(){
            tableAutores.setSizeFull();
            tableAutores.setSelectable(true);
            tableAutores.setImmediate(true);

            tableAutores.addValueChangeListener(new
Property.ValueChangeListener() {
                  public void valueChange(ValueChangeEvent event) {
                        Object autorId = tableAutores.getValue();
                        if (autorId != null){

      formFieldGroup.setItemDataSource(tableAutores.getItem(autorId));
                              deleteButton.setEnabled(true);
                              editorLayout.setVisible(true);
                              Autor autor =
formFieldGroup.getItemDataSource().getBean();
                              if(autor.getImagenUrl()!=null &&
!autor.getImagenUrl().equalsIgnoreCase("")){
                                    ExternalResource resource = new
ExternalResource(autor.getImagenUrl());
```

```
                                        fotoAutor.setSource(resource);
                                        fotoAutor.setVisible(true);
                                  }else{
                                        fotoAutor.setVisible(false);
                                  }
                                  obraBeanItemContainer.removeAllItems();

            obraBeanItemContainer.addAll(autor.getObras());
                                  if(obraBeanItemContainer.size()>0){
                                        tableObras.setVisible(true);
                                  }else{
                                        tableObras.setVisible(false);
                                  }
                            }else{
                                  deleteButton.setEnabled(false);
                                  editorLayout.setVisible(false);
                            }
                            if(flagNuevo){

            autorBeanItemContainer.removeItem(newAutor);
                            }
                      }
                });
          }

          @SuppressWarnings("serial")
          private void buildButtons(){
                addButton = new Button("Añadir", new Button.ClickListener() {

                      @Override
                      public void buttonClick(ClickEvent event) {
                            newAutor = new Autor();
                            formFieldGroup.setItemDataSource(newAutor);
                            tableAutores.setValue(null);
                            editorLayout.setVisible(true);
                            fotoAutor.setVisible(false);
                            tableObras.setVisible(false);
                            flagNuevo = true;
                            deleteButton.setEnabled(false);
                      }
                });
                addButton.setDescription("Añadir autor");
                addButton.setStyleName(Reindeer.BUTTON_SMALL);

                deleteButton = new Button("Borrar", new
Button.ClickListener() {

                      @Override
                      public void buttonClick(ClickEvent event) {
                            try {

            autorService.deleteAutor(((Autor)tableAutores.getValue()));

            autorBeanItemContainer.removeItem(tableAutores.getValue());
                                  tableAutores.setValue(null);
                            } catch (Exception e) {
```

```
                                Notification.show("Error",
Notification.Type.ERROR_MESSAGE);
                                e.printStackTrace();
                        }

                }
        });
        deleteButton.setDescription("Borrar autor seleccionado");
        deleteButton.setStyleName(Reindeer.BUTTON_SMALL);
        deleteButton.setEnabled(false);
    }

    @SuppressWarnings("serial")
    private void buildEditor(){
        editorLayout = new HorizontalLayout();
        editorLayout.setSizeFull();

        form.setMargin(true);
        for(Entry<String,String> entry : formFields.entrySet()){
            TextField field = new TextField(entry.getValue());
            field.setNullRepresentation("");
            form.addComponent(field);
            field.setWidth("100%");

            formFieldGroup.bind(field, entry.getKey());
        }

        formFieldGroup.addCommitHandler(new
FieldGroup.CommitHandler() {
            @SuppressWarnings("unchecked")
            @Override
            public void postCommit(CommitEvent commitEvent) throws
CommitException {
                try {
                    Autor autor =
((BeanFieldGroup<Autor>)commitEvent.getFieldBinder()).getItemDataSource().
getBean();
                    if(flagNuevo){
                        autor=autorService.addAutor(autor);
                        newAutor=autor;
                    }else{
                        autorService.updateAutor(autor);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                    throw new CommitException();
                }
            }

            @Override
            public void preCommit(CommitEvent commitEvent) throws
CommitException {
            }
        });

        commit = new Button("Guardar", new Button.ClickListener() {
            @Override
```

```
                    public void buttonClick(ClickEvent event) {
                        try {
                            formFieldGroup.commit();
                            Notification.show("Guardado",
Notification.Type.TRAY_NOTIFICATION);
                            if(flagNuevo){

        autorBeanItemContainer.addBean(newAutor);
                                tableAutores.setValue(null);
                                editorLayout.setVisible(false);
                                flagNuevo=!flagNuevo;
                            }
                        } catch (CommitException e) {
                            Notification.show("Error",
Notification.Type.ERROR_MESSAGE);
                            e.printStackTrace();
                        }
                    }
                });
                discard = new Button("Cancelar", new Button.ClickListener() {
                    @Override
                    public void buttonClick(ClickEvent event) {
                        formFieldGroup.discard();
                        if(flagNuevo){
                            editorLayout.setVisible(false);
                            flagNuevo=!flagNuevo;
                        }
                    }
                });
                HorizontalLayout formButtons = new HorizontalLayout(commit,
discard);
                form.addComponent(formButtons);

                editorLayout.addComponent(form);
                editorLayout.addComponent(fotoAutor);
                editorLayout.addComponent(tableObras);

                fotoAutor.setVisible(false);
                tableObras.setVisible(false);

                editorLayout.setComponentAlignment(fotoAutor,
Alignment.MIDDLE_CENTER);
                editorLayout.setSpacing(true);

                tableObras.setSizeFull();

                fotoAutor.setHeight(100, Unit.PERCENTAGE);

                editorLayout.setVisible(false);
        }

}
```

E.2. ObraEditor.java

```
package es.lta.biblioteca.admin.components.impl;

import java.util.ArrayList;
```

```java
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.data.fieldgroup.BeanFieldGroup;
import com.vaadin.data.fieldgroup.FieldGroup;
import com.vaadin.data.fieldgroup.FieldGroup.CommitEvent;
import com.vaadin.data.fieldgroup.FieldGroup.CommitException;
import com.vaadin.data.util.BeanItemContainer;
import com.vaadin.shared.ui.combobox.FilteringMode;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.AbstractSelect.ItemCaptionMode;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.ComboBox;
import com.vaadin.ui.FormLayout;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.Notification;
import com.vaadin.ui.Table;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.VerticalSplitPanel;
import com.vaadin.ui.themes.Reindeer;

import es.lta.biblioteca.admin.components.EditorInterface;
import es.lta.biblioteca.admin.service.AutorService;
import es.lta.biblioteca.admin.service.ObraService;
import es.lta.biblioteca.admin.service.impl.AutorServiceImpl;
import es.lta.biblioteca.admin.service.impl.ObraServiceImpl;
import es.lta.biblioteca.model.Autor;
import es.lta.biblioteca.model.Etiqueta;
import es.lta.biblioteca.model.Obra;
import es.lta.biblioteca.utils.FormatUtils;

public class ObraEditor extends AbsoluteLayout implements EditorInterface{

	/**
	 * serialVersionUID autogenerado
	 */
	private static final long serialVersionUID = -8023806816373684727L;

	private ObraService obraService = new ObraServiceImpl();
	private AutorService autorService = new AutorServiceImpl();

	private BeanItemContainer<Obra> obraBeanItemContainer;
	private Table tableObras;
	private Button addButton;
	private Button deleteButton;

	private Button commit;
```

```
	private Button discard;

	private FormLayout form = new FormLayout();
	private BeanFieldGroup<Obra> formFieldGroup = new
BeanFieldGroup<Obra>(Obra.class);

	private String[] tableObrasFields = new String[] { "idObra",
"titulo", "autors", "etiquetas" };
	private String[] tableObrasHeaders = new String[] {"id", "Título",
"Autores", "Etiquetas"};

	private Map<String,String> formFields = new LinkedHashMap<String,
String>();

	private boolean flagNuevo = false;
	private Obra newObra;

	private HorizontalLayout editorLayout;

	private VerticalLayout layoutAutores;
	private BeanItemContainer<AutorWrapper> autorBeanItemContainer;

	public ObraEditor(){
		initFormFields();
		setSizeFull();
		initContainer();
		buildView();
	}

	@Override
	public void setSizeFull() {
		 setWidth(100, Unit.PERCENTAGE);
	    setHeight(100, Unit.PERCENTAGE);
	}

	@SuppressWarnings("serial")
	protected void initContainer(){
		obraBeanItemContainer = new
BeanItemContainer<Obra>(Obra.class, obraService.getAll());
		tableObras = new Table("Obras", obraBeanItemContainer){
			@SuppressWarnings("unchecked")
			@Override
			protected String formatPropertyValue(Object rowId,
Object colId,
					Property<?> property) {
				String pid = (String)colId;
				if(pid.equals("autors")){
					Set<Autor> autors =
(Set<Autor>)property.getValue();
					StringBuilder strBuilder = new
StringBuilder();
					Iterator<Autor> iter = autors.iterator();
					Autor autor;
					while(iter.hasNext()){
						autor = iter.next();
```

```
		strBuilder.append(FormatUtils.formatAutorAsN_A_A(autor));
                                        if(iter.hasNext()){
                                                strBuilder.append(", ");
                                        }
                                }
                                return strBuilder.toString();
                        }else if(pid.equals("etiquetas")){
                                Set<Etiqueta> etiquetas =
(Set<Etiqueta>)property.getValue();
                                StringBuilder strBuilder = new
StringBuilder();
                                Iterator<Etiqueta> iter =
etiquetas.iterator();
                                Etiqueta etiqueta;
                                while (iter.hasNext()) {
                                        etiqueta = iter.next();


		strBuilder.append(FormatUtils.formatEtiquetaAsC_PP_E(etiqueta));
                                        if(iter.hasNext()){
                                                strBuilder.append(", ");
                                        }
                                }
                                return strBuilder.toString();
                        }
                        return super.formatPropertyValue(rowId, colId,
property);
                }
        };

        List<AutorWrapper> autores = new ArrayList<AutorWrapper>();
        AutorWrapper autorWrapper;
        for(Autor autor : autorService.getAll()){
                autorWrapper = new AutorWrapper();
                autorWrapper.setIdAutor(autor.getIdAutor());

		autorWrapper.setNombre(FormatUtils.formatAutorAsN_A_A(autor));
                autores.add(autorWrapper);
        }
        Collections.sort(autores);

        autorBeanItemContainer = new
BeanItemContainer<AutorWrapper>(AutorWrapper.class, autores);
        tableObras.setVisibleColumns(tableObrasFields);
        tableObras.setColumnHeaders(tableObrasHeaders);
	}

	public void reView(){
        editorLayout.setVisible(false);

        tableObras.setValue(null);

        obraBeanItemContainer.removeAllItems();
        obraBeanItemContainer.addAll(obraService.getAll());

        autorBeanItemContainer.removeAllItems();
```

```
		List<AutorWrapper> autores = new ArrayList<AutorWrapper>();
		AutorWrapper autorWrapper;
		for(Autor autor : autorService.getAll()){
			autorWrapper = new AutorWrapper();
			autorWrapper.setIdAutor(autor.getIdAutor());

	autorWrapper.setNombre(FormatUtils.formatAutorAsN_A_A(autor));
			autores.add(autorWrapper);
		}
		Collections.sort(autores);
		autorBeanItemContainer.addAll(autores);
	}

	private void initFormFields(){
		formFields.put("titulo", "Título");
	}

	private void buildView(){
		setSizeFull();

		VerticalSplitPanel verticalSplitPanel = new
VerticalSplitPanel();
		verticalSplitPanel.setSizeFull();

		addComponent(verticalSplitPanel);

		buildTableObras();
		verticalSplitPanel.addComponent(tableObras);

		buildButtons();
		addComponent(addButton, "top:0;right:70px;");
		addComponent(deleteButton, "top:0;right:5px;");

		buildEditor();

		verticalSplitPanel.addComponent(editorLayout);
	}

	@SuppressWarnings("serial")
	private void buildTableObras(){
		tableObras.setSizeFull();
		tableObras.setSelectable(true);
		tableObras.setImmediate(true);

		tableObras.addValueChangeListener(new
Property.ValueChangeListener() {

			@Override
			public void valueChange(ValueChangeEvent event) {
				Object obraId = tableObras.getValue();
				if(obraId!=null){

	formFieldGroup.setItemDataSource(tableObras.getItem(obraId));
					deleteButton.setEnabled(true);
					editorLayout.setVisible(true);
					buildLayoutAutores();
				}else{
```

```
                                deleteButton.setEnabled(false);
                                editorLayout.setVisible(false);
                        }
                        if(flagNuevo){
                                obraBeanItemContainer.removeItem(newObra);
                        }
                }
        });
    }

    @SuppressWarnings("serial")
    private void buildButtons(){
        addButton = new Button("Añadir", new Button.ClickListener() {

                @Override
                public void buttonClick(ClickEvent event) {
                        newObra = new Obra();
                        formFieldGroup.setItemDataSource(newObra);
                        tableObras.setValue(null);
                        layoutAutores.removeAllComponents();
                        editorLayout.setVisible(true);
                        flagNuevo = true;
                        deleteButton.setEnabled(false);
                }
        });
        addButton.setDescription("Añadir obra");
        addButton.setStyleName(Reindeer.BUTTON_SMALL);

        deleteButton = new Button("Borrar", new
Button.ClickListener() {

                @Override
                public void buttonClick(ClickEvent event) {
                        try {

    obraService.deleteObra(((Obra)tableObras.getValue()));

    obraBeanItemContainer.removeItem(tableObras.getValue());
                                tableObras.setValue(null);
                        } catch (Exception e) {
                                Notification.show("Error",
Notification.Type.ERROR_MESSAGE);
                                e.printStackTrace();
                        }
                }
        });
        deleteButton.setDescription("Borrar obra seleccionada");
        deleteButton.setStyleName(Reindeer.BUTTON_SMALL);
        deleteButton.setEnabled(false);
    }

    @SuppressWarnings("serial")
    private void buildEditor(){
        editorLayout = new HorizontalLayout();
        editorLayout.setSizeFull();

        form.setMargin(true);
```

```
            for(Entry<String,String> entry : formFields.entrySet()){
                  TextField field = new TextField(entry.getValue());
                  field.setNullRepresentation("");
                  form.addComponent(field);
                  //field.setWidth("100%");

                  formFieldGroup.bind(field, entry.getKey());
            }

            formFieldGroup.addCommitHandler(new
FieldGroup.CommitHandler() {
                  @SuppressWarnings("unchecked")
                  @Override
                  public void postCommit(CommitEvent commitEvent) throws
CommitException {
                        try {
                              Obra obra =
((BeanFieldGroup<Obra>)commitEvent.getFieldBinder()).getItemDataSource().g
etBean();
                              if(flagNuevo){
                                    obra=obraService.addObra(obra);
                                    newObra=obra;
                              }else{
                                    obraService.updateObra(obra);
                              }
                        } catch (Exception e) {
                              e.printStackTrace();
                              throw new CommitException();
                        }
                  }

                  @Override
                  public void preCommit(CommitEvent commitEvent) throws
CommitException {
                  }
            });

            commit = new Button("Guardar", new Button.ClickListener() {

                  @Override
                  public void buttonClick(ClickEvent event) {
                        try {
                              formFieldGroup.commit();
                              Notification.show("Guardado",
Notification.Type.TRAY_NOTIFICATION);
                              if(flagNuevo){

      obraBeanItemContainer.addBean(newObra);
                                    editorLayout.setVisible(false);
                                    flagNuevo=!flagNuevo;
                              }
                        } catch (CommitException e) {
                              Notification.show("Error",
Notification.Type.ERROR_MESSAGE);
                              e.printStackTrace();
                        }
                  }
```

```
		} );
		discard = new Button("Cancelar", new Button.ClickListener() {
			@Override
			public void buttonClick(ClickEvent event) {
				formFieldGroup.discard();
				if(flagNuevo){
					editorLayout.setVisible(false);
					flagNuevo=!flagNuevo;
				}
			}
		});

		layoutAutores = new VerticalLayout();
		form.addComponent(layoutAutores);

		HorizontalLayout formButtons = new HorizontalLayout(commit,
discard);
		form.addComponent(formButtons);

		editorLayout.addComponent(form);

		editorLayout.setVisible(false);
	}

	@SuppressWarnings("serial")
	private void buildLayoutAutores(){
		Obra obra = formFieldGroup.getItemDataSource().getBean();
		layoutAutores.removeAllComponents();

		ComboBox comboBox = new ComboBox("Autores",
autorBeanItemContainer);
		comboBox.setInputPrompt("Añadir");
		comboBox.setItemCaptionPropertyId("nombre");
		comboBox.setItemCaptionMode(ItemCaptionMode.PROPERTY);
		comboBox.setFilteringMode(FilteringMode.CONTAINS);
		comboBox.setImmediate(true);
		comboBox.setNullSelectionAllowed(false);

		comboBox.addValueChangeListener(new
Property.ValueChangeListener() {

			@Override
			public void valueChange(ValueChangeEvent event) {
				AutorWrapper autorWrapper =
(AutorWrapper)event.getProperty().getValue();
				Autor autor =
autorService.getAutor(autorWrapper.getIdAutor());
				Obra obra =
formFieldGroup.getItemDataSource().getBean();
				obra.getAutors().add(autor);
				Notification.show("Autor añadido",
Notification.Type.TRAY_NOTIFICATION);
				buildLayoutAutores();
			}
		});

		layoutAutores.addComponent(comboBox);
```

```
            for(Autor autor : obra.getAutors()){
                  HorizontalLayout autorLayout = new HorizontalLayout();
                  autorLayout.addComponent(new
Label(FormatUtils.formatAutorAsN_A_A(autor)));

                  Button button = new Button("Quitar");
                  button.setStyleName(Reindeer.BUTTON_SMALL);
                  button.setData(autor);
                  button.addClickListener(new Button.ClickListener() {

                        @Override
                        public void buttonClick(ClickEvent event) {
                              Autor autor =
(Autor)event.getButton().getData();
                              Obra obra =
formFieldGroup.getItemDataSource().getBean();
                              obra.getAutors().remove(autor);
                              Notification.show("Autor quitado",
Notification.Type.TRAY_NOTIFICATION);
                              buildLayoutAutores();
                        }
                  });
                  autorLayout.addComponent(button);
                  autorLayout.setSpacing(true);
                  layoutAutores.addComponent(autorLayout);
            }
            layoutAutores.setSpacing(true);
      }

      public class AutorWrapper implements Comparable<AutorWrapper>{

            private Integer idAutor;
            private String nombre;

            public Integer getIdAutor() {
                  return idAutor;
            }

            public void setIdAutor(Integer idAutor) {
                  this.idAutor = idAutor;
            }

            public String getNombre() {
                  return nombre;
            }

            public void setNombre(String nombre) {
                  this.nombre = nombre;
            }

            public int compareTo(AutorWrapper other) {
                  AutorWrapper otroAutor = (AutorWrapper) other;
                  return this.nombre.compareTo(otroAutor.getNombre());
            }
      }
}
```

Anexo F: Documento de Objetivos del Proyecto (DOP)

A continuación anexamos el DOP confeccionado al comienzo de este PFC.

Motivación del proyecto

Son dos aspectos los que motivan la propuesta de este proyecto, otorgándole al mismo un doble carácter. El primero, en el plano funcional, surge por la necesidad real de gestionar eficientemente el catálogo de una abultada librería doméstica; lo que imprime al proyecto su carácter útil.

El segundo, en el plano técnico, impregna de un carácter formativo que viene a afianzar lo aprendido en los anteriores años de estudio o vida profesional, en algunos casos; y a introducir en el aprendizaje de otras materias, en otros. Este segundo aspecto, más apetitoso en lo personal, es del que esperamos sacar mayor provecho.

Escenario actual

Nuestra librería doméstica consta de un grueso aproximado de unos 700 volúmenes, con un crecimiento anual estimado de 60-80 volúmenes. Se trata de libros de diversos temas, no necesariamente conexos, que se encuentran en varios idiomas. En algunos casos, la misma obra puede encontrarse en distintas ediciones. En muy pocos casos existen varias copias de una misma edición. Todos los ejemplares se almacenan en un mismo espacio físico.

Además, se dispone de un, por el momento, pequeño catálogo de libros electrónicos (*ebook*) de también diverso temario, etc., en diversos formatos. Algunos *ebooks* están disponibles en más de un formato.

Requerimientos

Se necesita un sistema de catalogación que permita una introducción de datos sencilla y un posterior acceso y modificación de los mismos igualmente sencillo. Los datos de cada volumen que han de ser registrados son, al menos, los siguientes:

- Título y autor de la obra, y su relación con la misma (autor, editor, etc.).
- Nombre de la editorial, año y ciudad de publicación.
- Número de edición, en caso de ser mayor a la 1ª

- ISBN[74] y/o EAN[75] (si los tuviere)

Del mismo modo, es requisito indispensable que el catálogo sea fácilmente categorizable, permitiendo definir los temas de los que trata cada uno de los libros. El sistema ha de proporcionar un método sencillo y flexible para la gestión de las categorías que se manejen.

El catálogo ha de ser posteriormente explorable mediante una interfaz gráfica, permitiendo realizar búsquedas en función de los criterios que indique el usuario. Estos criterios serán relativos a los datos almacenados para cada libro.

Estos requisitos aplican tanto para los volúmenes físicos como electrónicos. La aplicación ha de especificar en qué soporte se encuentra el ejemplar (físico, electrónico/digital, o los dos). No es requisito, pero sí es deseable, facilitar el acceso a la versión digital, si la tuviere, de un volumen.

Propuesta funcional del proyecto

Para responder a estas necesidades se propone la construcción de un sistema que permita al usuario: 1) registrar, consultar, modificar y borrar la información solicitada de los libros; 2) gestionar un sistema de etiquetas que proporcione un método de categorización y meta-información a los volúmenes; 3) proporcionar una herramienta de búsqueda a partir de las etiquetas que especifique el usuario; y 4) permitir la navegación por el catálogo en base a los criterios que especifique el usuario mediante las etiquetas que se encuentren en el sistema.

Este sistema será accesible a través de un navegador web y desde una aplicación para dispositivos móviles Android.

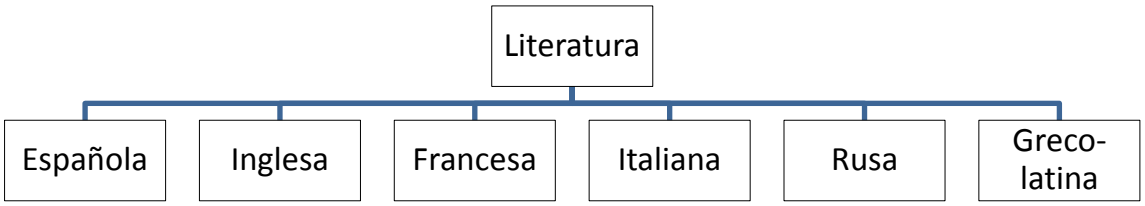
Las etiquetas tienen a su vez categorías, de modo que posteriormente podremos definir nuevas categorías para nuevas etiquetas. Inicialmente se proponen seis categorías de etiquetas: Autor, Editorial, Género, Época, Tema e Idioma. Las etiquetas se ordenan en forma de árbol, de modo que estas vayan de lo más general a lo más específico. Por ejemplo:

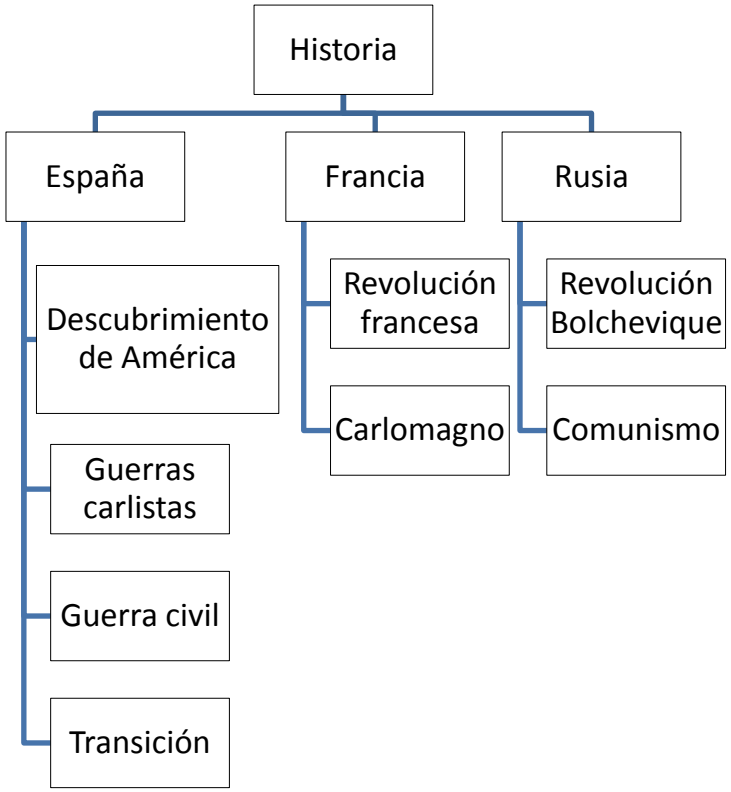
---

[74] *International Standard Book Number*, Número Estándar Internacional de Libro. Es un identificador único para libros, previsto para uso comercial.

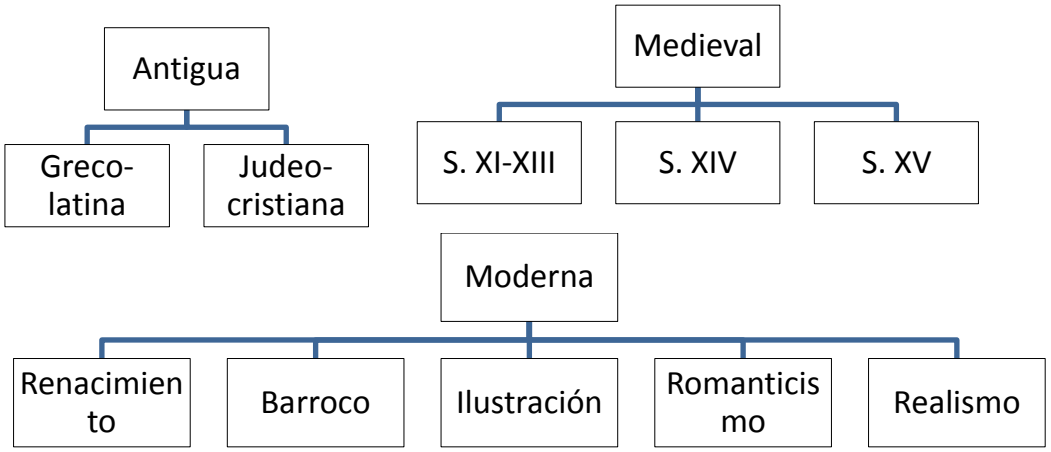
[75] *European Article Number*, Número Europeo de Artículo. Es un sistema de códigos de barras. Desde enero de 2007, el sistema EAN-13 se compatibiliza con el ISBN-13 en España. http://www.mcu.es/libro/CE/AgenciaISBN/InfGeneral/ISBN13.html

Tema


Literatura
Española
Inglesa
Francesa
Italiana
Rusa
Greco-latina


Historia
España
Francia
Rusia
Descubrimiento de América
Guerras carlistas
Guerra civil
Transición
Revolución francesa
Carlomagno
Revolución Bolchevique
Comunismo

Época


Antigua
Greco-latina
Judeo-cristiana
Medieval
S. XI-XIII
S. XIV
S. XV
Moderna
Renacimiento
Barroco
Ilustración
Romanticismo
Realismo

La gestión del sistema de etiquetas permitirá la creación, modificación, consulta y borrado tanto de categorías como de etiquetas, así como la búsqueda y la navegación por ellas.

Esbozo arquitectónico

Para la consecución de este proyecto se propone la construcción de un sistema por capas, compuesto por una capa de persistencia de datos, una capa de lógica de negocio y una tercera capa de presentación. A menos que se indique lo contrario, la programación será en lenguaje Java.

Estas capas estarán implementadas de la siguiente manera:

1. Capa de persistencia: Destinada al almacenamiento de la información que maneje el sistema. El almacenamiento físico correrá a cargo de un SGBD[76], preferiblemente Oracle[77] o MySQL[78]. Por encima del SGBD realizaremos un mapeo objeto-relacional con un ORM[79], preferiblemente Hibernate[80], que nos permitirá la persistencia de los objetos y definirá las interfaces CRUD[81] para la explotación básica de los mismos. Con la intención de no hacer el sistema dependiente del ORM elegido, se definirán interfaces DAO[82] que nos proporcionarán las funciones de persistencia que previamente haya inyectado Spring[83].

2. Lógica de negocio: Responsable de la explotación de los datos. Se programará modularmente, en base a los casos de uso que implemente el sistema, mediante el desarrollo de unos servicios web REST en JAX-RS[84], especificación presente en la versión EE 6 de JAVA. La construcción de estos servicios web hará que el acceso a la lógica de negocio sea independiente de la ubicación, plataforma o lenguaje del cliente que lo consume. Serán implementados con el *framework* RESTEasy[85].

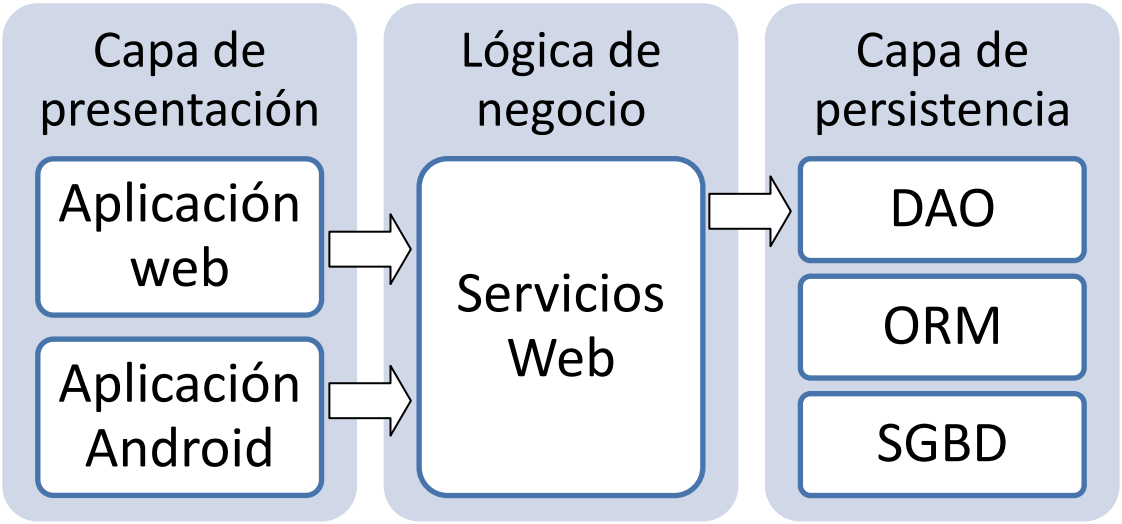
---

[76] Sistema de Gestión de Bases de Datos
[77] En su versión gratuita *Express Edition*. http://www.oracle.com/technetwork/database/express-edition/overview/index.html.
[78] En su versión gratuita *Community Server*. http://www.mysql.com/products/community/.
[79] Object-Relational Mapping
[80] Herramienta ORM ampliamente usada en las plataformas Java
[81] *Create Read Update Delete*, Crear Leer Actualizar Borrar
[82] *Data Access Object*, Objeto de Acceso a Datos
[83] Dentro de las posibilidades del framework Spring, utilizaremos su característica principal de IoC (*Inversion of Control*, Inversión de Control) para la inyección de dependencias. http://www.springsource.com
[84] *JAX-RS: Java API for RESTful Web Services*, API de Java que permite la creación de servicios web de acuerdo a la técnica REST (*Representational State Transefer*, Transferencia de Estado Representacional). Ver JSR-311 para más detalles: http://jcp.org/en/jsr/detail?id=311
[85] Implementación de la especificación JAX-RS de JBoss.

3. Capa de presentación: Como se ha expuesto anteriormente, el acceso por parte del usuario al sistema se efectuará, bien a través de una interfaz web, bien mediante una aplicación Android. Para esta última se utilizarán las herramientas de desarrollo proporcionadas por Google[86].

   En el caso de la aplicación web, se implementará con el *framework* GWT, también de Google. Es deseable que la interfaz resultante sea vistosa y usable, y que proporcione un flujo rápido en las acciones. Es decir, se desea una aplicación web RIA[87], que haga uso intensivo de AJAX[88].

Gráficamente, podemos representar este diseño en capas de la siguiente manera:



Este diseño por capas permite afrontar la construcción del proyecto por módulos, estableciendo entre ellos un bajo acoplamiento. Fruto de este bajo acoplamiento, un cambio en el alcance por añadidura de operaciones no comporta un cambio traumático en el sistema. Puede bastar, por ejemplo, con añadir un nuevo servicio web con nueva lógica para explotar en otro orden los datos sin verse afectadas las capas de persistencia ni las capas de presentación (salvo para incorporar las nuevas funcionalidades de este servicio web).

La construcción del proyecto por módulos también nos permitirá centrarnos cada vez en la construcción de un módulo en concreto, evaluando de manera individual su diseño y afrontando su programación independientemente de la del resto. Así, identificamos varios módulos a construir:

---

[86] Básicamente, la SDK (*Software Development Kit*). http://developer.android.com/sdk/index.html
[87] *Rich Internet Applications*, término que se utiliza para denominar aquellas aplicaciones web que proporcionan características avanzadas e interactivas al usuario y que visualmente se presentan mucho más ricas que las aplicaciones web tradicionales.
[88] *Asynchronous Javascript And XML*, es una técnica de desarrollo web para aplicaciones RIA.

1. Modelo de datos para la capa de persistencia y elaboración de los DAO.
2. Programación de los servicios web.
3. Creación de la aplicación Android
4. Creación de la aplicación web.

Metodología de trabajo

En cuanto a la programación del sistema, ésta se realizará con Eclipse[89] como IDE[90]. En el caso de que alguno de los desarrollos comportara o recomendara el cambio a otro IDE se documentaría pertinentemente. La compilación del código JAVA se hará con la última versión de su JDK[91], actualmente la versión 7. Si, igualmente, alguno de los desarrollos hiciera necesario el uso de alguna versión anterior se indicará en la documentación correspondiente.

Se afrontará el desarrollo de los módulos en el orden citado en el apartado anterior, es decir, comenzando por la persistencia de datos y terminando por la aplicación web. Al inicio de cada módulo se realizará el estudio pertinente de cara a implementarlo de la manera que convenga.

Cada uno de los módulos contará con su propia documentación, compuesta por análisis funcional, diseño técnico y memoria de desarrollo, de modo que todo el proceso de construcción del módulo quede precisamente documentado.

Todo el desarrollo se registrará en un sistema de control de versiones. De este modo dispondremos de todo el código centralizado, accesible desde distintas ubicaciones (aunque se utilizará sólo una) y en sus distintas versiones. El sistema que se utilizará será Subversion[92] (en adelante SVN).

Para el seguimiento y la gestión del proyecto se utilizará la herramienta JIRA[93], que permite la creación de tareas, flujos de trabajo e imputación de horas dedicadas. También es posible enlazar JIRA con el servidor de SVN de modo que se asocie cada una de las tareas con el código que le corresponde.

Todo el desarrollo se realizará en el equipo del alumno. Las pruebas y puestas en marcha en entornos pre-productivos se realizarán en máquinas virtuales alojadas en el mismo

---

[89] http://www.eclipse.org/
[90] *Integrated Development Environment*, Entorno de Desarrollo Integrado
[91] *Java Development Kit*
[92] http://subversion.apache.org/
[93] http://www.atlassian.com/jira

ordenador. Los servidores de SVN y JIRA se alojarán también en una máquina virtual. El software para la gestión de máquinas virtuales será Virtual Box[94] de Oracle.

Finalmente, y dado que el alumno no reside cerca del centro universitario, el seguimiento del proyecto por parte del tutor del centro se realizará vía correo electrónico. Cuando la situación lo permita se llevarán a cabo reuniones presenciales, estando la primera ya concertada para la semana del 17 de diciembre de 2012.

Hitos del proyecto

Las fechas estimativas de finalización para las tareas principales son las siguientes:

1.	Redacción del presente documento	24 de septiembre de 2012
2.	Configuración del entorno de gestión (SVN+JIRA)	30 de septiembre de 2012
3.	Módulo de datos	31 de octubre de 2012
4.	Módulo de servicios web	31 de diciembre de 2012
5.	Módulo de aplicación Android	31 de marzo de 2013
6.	Módulo de aplicación web	20 de mayo de 2013
7.	Elaboración de memoria	31 de mayo de 2013

Obviamente estas tareas se descomponen en otras de grano más fino. Sin embargo, una planificación más detallada resultaría, a todas luces, del todo irreal. Del mismo modo, hemos desestimado la valoración de las tareas en horas, principalmente porque una planificación inicial busca marcar hitos en el desarrollo del proyecto más que establecer una exhaustiva planificación a largo plazo.

La gestión continua del proyecto mediante la herramienta JIRA nos permitirá llevar un control diario y un registro preciso de las tareas. La planificación se realizará de manera dinámica, siempre marcando objetivos a corto plazo, asignando como próxima tarea aquella que convenga por orden o prioridad de entre las tareas que se encuentren abiertas y registradas en el JIRA. Finalmente, al finalizar el proyecto, la herramienta nos proporcionará el registro detallado de tareas realizadas y la dedicación de cada una de ellas.

La dedicación diaria del alumno al proyecto resultará variable debido a sus responsabilidades laborales y familiares. En cualquier caso, y de manera orientativa, supondrán unas 12-16 horas semanales.

[94] https://www.virtualbox.org/

Anexo G: Bibliografía y recursos online

Ableson, W. F.; Sen, R. & King, C. (2011). *Android, Guía para desarrolladores*, 2ª edición, Madrid: Anaya (Manning)

Burke, B. (2010). *RESTful Java with JAX-RS*, Sebastopol, CA: O'Reilly

Casey, J.; Massol, V.; Porter, B.; Sánchez, C. & Van Zyl, J. (2008). *Better builds with Maven*, Exist Global[95]

Debrauwer, L. & Van der Heyde, F. (2009). *UML 2*, Cornellà de Llobregat, Barcelona: Eni Ediciones.

Grönroos, M. (2013). *Book of Vaadin: Vaadin 7 Edition*, https://vaadin.com/book

Hightower, R. (2011). *CDI Dependency Injection*, http://java.dzone.com/articles/cdi-di-p1

JBoss. *Hibernate Tools Reference Guide, Version 3.3.1 GA*, http://www.jboss.org/tools/docs

Lima, F. (2009). *Java 6*, Madrid: Anaya

[95] Disponible bajo licencia CC en http://www.maestrodev.com/better-builds-with-maven/about-this-guide/

Anexo H: contenidos del DVD que acompaña a la memoria

El DVD adjunto a esta memoria incluye todo el código fuente del proyecto, así como la máquina virtual utilizada y otros elementos relevantes del proyecto. Están estructurados de la siguiente manera.

- db/
  Incluye el script de creación de base de datos, una imagen con el diagrama EER y un fichero *.mwb con el diseño de base de datos que puede abrirse con MySQL Workbench.
- mv/
  Incluye la máquina virtual, en formato VirtualBox, que se ha utilizado en el proyecto. Los usuarios de la máquina y los detalles de la instalación están indicados en el anexo correspondiente.
- servers/
  En esta carpeta se encuentran los servidores Tomcat y TomEE utilizados para realizar los despliegues de los desarrollos.
- src/
  Aquí encontramos todo el código escrito para este PFC. Puede importarse en Eclipse.

Además, en la raíz del DVD se encuentra una copia de esta memoria.