UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Migración de un catálogo y repositorio de imágenes de máquinas virtuales a Java/Hibernate

Proyecto Final de Carrera

Ingeniería Informática

Autor: Ángel García Villalobos

Director: Germán Moltó Martínez

07/09/12

Resumen

Últimamente el entorno empresarial se ha dado cuenta de la gran ventaja de utilizar entornos virtualizados, gracias a la fácil configuración, mantenimiento y disposición de los mismos. Para ello es recomendable apoyarse en algún mecanismo para gestionar el gran número de imágenes de máquinas virtuales que se posean.

El presente proyecto trata justamente del desarrollo de una aplicación con arquitectura cliente/servidor (en concreto de la parte del servidor) que ofrezca servicios de catálogo de imágenes de máquinas virtuales, el cual facilita las labores de mantenimiento (altas, búsquedas, o modificaciones) de dichas virtualizaciones.

A la vez que se desarrolla una funcionalidad especifica, el proyecto intenta ir más allá y realizar un estudio de marcos y metodología de trabajo para conseguir una aplicación bien estructurada que sea fácilmente ampliable incluso por desarrolladores diferentes al inicial.

Agradecimientos

El autor desea expresar su más sincero agradecimiento al profesor Germán Moltó Martínez por su colaboración tanto en la preparación de este manuscrito como en el desarrollo de cada una de las partes del proyecto, a los compañeros de empresa por el tiempo y conocimientos compartidos. Gracias también a la familia por su incondicional apoyo. Y por último gracias, Marta pues eres la que me ha dado fuerzas para continuar en todo momento.

Tabla de contenidos

1. Introducción

El presente Proyecto Final de Carrera se enmarca en una colaboración con el Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) de la Universitat Politècnica de València. El objetivo consistía en desarrollar un servicio de gestión de imágenes de máquinas virtuales como evolución de un software ya existente llamado VMRC (Virtual Machine image Repository & Catalog). Se pretende que el nuevo desarrollo realice una mejor gestión de los datos mediante el uso de tecnología ORM (Object Relational Mapping) concretada en el software Hibernate.

Vamos a comenzar a explicar cada una de las partes que componen el proyecto VMRC, tanto las características técnicas, como las ventajas que su uso conllevará.

1.1 Motivación

Este proyecto trata del desarrollo de una aplicación de gestión de un catálogo de imágenes de máquinas virtuales. La motivación de este proyecto tiene dos vertientes. Una es la importancia hoy en día de la virtualización, y otra es la de una metodología o pautas de desarrollo de aplicaciones cuyo fin es tener un código lo más desacoplado posible de la estructura de la base de datos.

1.1.1 Virtualización

La virtualización ha tomado una gran importancia en diferentes campos tanto profesionales como académicos. La idea es permitir tener en una misma máquina instancias de otros sistemas, ya sea emulando un determinado hardware y/o software. La gestión de este segundo sistema se realiza mediante los llamados Hipervisores o monitores de máquinas virtuales. Por poner un ejemplo, puedes estar ejecutando Windows en tu ordenador, y tener un Hipervisor ejecutando una instancia de Linux de manera que para ese Linux su único contexto es él mismo.

Esto tiene grandes ventajas. En el contexto académico puedes probar varios sistemas operativos sin tener que tenerlos instalados físicamente en tu disco de arranque, o incluso tener sistemas operativos de otras arquitecturas que no sean X86 sin tener que salir de tu sistema. Por parte empresarial, desde un punto de vista de desarrollo o mantenimiento de software, permite tener instancias que simulen las diferentes instalaciones que puedan tener sus clientes, con el ahorro de hardware y costes de administración que eso conllevaría.

Dado este auge de la virtualización nos encontramos con que en determinados contextos el número de imágenes de máquinas virtuales puede llegar a ser lo suficientemente grande (pensar por ejemplo en una universidad que tengan una máquina virtual configurada por cada asignatura que se imparta...) como para que se

necesiten herramientas de gestión de dicha colección de máquinas virtuales. Dicha herramienta debería permitir al menos las acciones más básicas de toda gestión de catálogos como añadir, listar, buscar, editar, etc.

1.1.2 Desarrollo mantenible

Se tiene que tener en cuenta que las aplicaciones rara vez se terminan y no se vuelven a tocar. De ahí la importancia que la aplicación sea fácilmente modificable. Ante un cambio de requisito funcional, cuantas menos líneas de código se tengan que cambiar menos esfuerzo supondrá al desarrollador.

Una parte muy importante en una aplicación es su modelo de dominio, que es la representación de la realidad a objetos en el sistema. Comúnmente esta representación se persiste en bases de datos lo que lleva a la necesidad de métodos o mecanismos para realizar un intercambio y transformación de información entre ésta y la aplicación. En este caso nos centraremos en la transformación de la información, es decir como pasar de datos en tablas relacionales, a clases de domino escritas esta vez en el lenguaje de programación Java.

1.2 Objetivos

El objetivo principal del proyecto es modificar una aplicación existente para que cumpla los nuevos requisitos y a su vez esté preparada de cara a que los nuevos requisitos sean fácilmente alcanzables.

También enseñar el uso de herramientas no estándares de un ecosistema como puede ser el lenguaje Java, para facilitar tareas en todo el ciclo del desarrollo de una aplicación.

1.3 Estructura de la Memoria

Esta memoria está dividida en 5 bloques principales.

En el **capítulo 1** se hace una introducción al ámbito de la memoria, explicando la importancia del tema tratado y cómo se abordará. En el **capítulo 2** se enumeran y explican las tecnologías y herramientas utilizadas para el desarrollo del proyecto. En el **capitulo 3** se adentra en el problema en cuestión, dando una visión global de la aplicación y explicando como se ha llevado a cabo la parte desarrollada en el proyecto, utilizando las tecnologías enumeradas en el capitulo 2, pero esta vez desde el punto de vista de su aplicación en el caso concreto. El **capítulo 4** es una recapitulación de conclusiones, puntos de vista, problemas y demás opiniones que le han surgido al autor a lo largo de todo el desarrollo del proyecto.

2. Conceptos previos y tecnologías empleadas

2.1 Lenguaje de programación: Java

Para el desarrollo propiamente dicho de la aplicación que hará de servidor de nuestra arquitectura se ha optado por el lenguaje Java.

Java es un lenguaje orientado a objetos, creado por la empresa Sun Microsystems en 1995 y actualmente mantenido por Oracle en su versión 7. Algunas de las características del lenguaje son las siguientes:

- Es un lenguaje interpretado, esto es que necesita de una máquina virtual para ejecutarse. El código (ficheros .java) se compila a bytecode (ficheros .class) que es lo que en tiempo real interpretará la máquina virtual de Java. La mayor ventaja de esto es la portabilidad de aplicaciones, es decir una misma aplicación funcionará en cualquier sistema operativo que disponga de una máquina virtual.

- Orientado a objetos. Utiliza toda la potencia de la programación orientada a objetos como es la herencia al extender clases o al implementar interfaces. A partir de la versión 5 aparecen los genéricos que permiten tener clases parametrizadas, y anotaciones que dan una funcionalidad semántica a diferentes recursos (clases, métodos, campos…)

- Autogestión de la memoria, llevada a cabo por el llamado *garbage collector* (recolector de basura), que se encarga de liberar recursos que se hayan dejado de utilizar durante la ejecución.

Se procede a explicar algunos frameworks importantes en el mundo de Java, o al menos los que han sido utilizados para el desarrollo de este proyecto.

2.1.1 Entorno de desarrollo: Eclipse

El proyecto está desarrollado en Eclipse, un *IDE* (Integrated Development Environment) para Java. Se elige Eclipse al ser uno de los entornos gratuitos más extendidos en el mundo Java. Originalmente desarrollado por IBM, en estos momentos cuenta con una fundación desarrolladora propia (Eclipse Foundation), una organización de la que forman parte muchas de las grandes empresas de software de hoy en día, como pueden ser Adobe, Google, HP, Oracle, etc… una lista que llega a más de 20 componentes.

Como entorno, ofrece facilidades para llevar a cabo tareas del desarrollo como pueden ser:

- **Gestión de proyectos:** Permite crear, importar, exportar proyectos de Java, y clasificarlos según su naturaleza (consola, web, entorno grafico…).

- **Integración de entornos:** Permite gestionar entornos servidores como pueden ser Jetti, Tomcat, OC4J, etc… Tanto una simple gestión de arrancar/parar servicios, pasando por el despliegue automático de proyectos, hasta un total control en el debug de una aplicación.

- **Escritura de código:** En este sentido son muy potentes sus opciones de:

    - Autocompletado de código: Opciones para empezar a escribir una sentencia y te autocomplete en función de qué Clase estás accediendo. A su vez interesante ya que inserta automáticamente las sentencias **import** cuando las necesita.

    - Renombrado: De variables, métodos y clases. Cambio de nombre a variables, métodos y clases modificando automáticamente todas sus referencias.

    - Rastrear las llamadas: Saber todos los puntos del código que están usando un método o accediendo a una variable en particular.

    - Movimiento y copia de bloques: Con un simple atajo se pueden mover o copiar bloques enteros de código.

    - Refactorizar: Entre otras cosas permite coger un trozo de código crear una función auxiliar conteniéndolo, y modificando todos las partes de código que estaban repetidos.

Actualmente existe un sinfín de extensiones que añaden potencia a sus funcionalidades. A partir de la penúltima versión (Indigo), existe un *market place* donde se pueden buscar dichos *plugins* más extendidos e instalarlos automáticamente (anteriormente dicha instalación funcionaba en base a la búsqueda de repositorios previamente establecidos).

2.1.2 Subversion

Es un *CVS* (sistema de control de versiones) que sustituye la anterior aplicación homónima al sistema CVS (Concurrent Versions System). En general estas herramientas tienen dos funciones principales. Permiten tanto la compartición de código por un grupo de trabajo, como la gestión de cambios que se realizan en el mismo.

La estructura es la siguiente, hay un repositorio central donde se alberga el código, y los clientes que se conectan a él para comprobar el estado del mismo. En el caso de subversion se controla todo en base a las llamadas revisiones, que es el número que

identifica un estado en la versión del código. Dentro de un repositorio pueden haber varios proyectos, y cada proyecto a su vez se estructura en tres apartados:

- **Trunk (o rama principal):** Pese a que la estructura suele ser estándar, su utilización puede cambiar en función de la metodología empleada. En algunos casos este apartado suele utilizarse para contener el código que se está desarrollando en un determinado momento. En otros casos únicamente contiene el código de la versión estable del sistema.

- **Branches (o ramas de desarrollo):** Se utiliza para abrir una rama alternativa al desarrollo principal. Hay quien lo utiliza como desarrollos de pruebas. También se puede utilizar para desarrollar algún correctivo de una versión anterior. Aunque hay quien lo usa siempre que comienza a desarrollar algo para dejar el trunk de manera estable.

- **Tags:** Se utiliza para etiquetar versiones, ya sea por completar alguna funcionalidad especifica, o bien por haber hecho alguna entrega de producto. Una regla que se suele seguir es que nunca se debe desarrollar en un tag, en todo caso se obtendrá código de uno para arreglar algún error y con el resultado se volverá a crear otro, pero nunca modificar uno existente.

2.1.3 Maven

Es una herramienta de gestión de proyectos Java, iniciada por Jakarta y continuada por Apache Software Foundation.

Facilita la mayoría de las tareas a lo largo del ciclo de vida de un proyecto, desde gestión de dependencias (librerías usadas por la aplicación), pasando por ejecución de tests unitarios hasta el empaquetado y despliegue de la aplicación.

Se gestiona utilizando un fichero POM (Project Object Model), que no es más que un XML de configuración con un formato determinado.

Entre sus funcionalidades o ventajas se encuentran:

- **Gestión de dependencias:** Normalmente ficheros JAR que contienen paquetes de código con una funcionalidad determinada. Maven ayuda a gestionarlos de manera que únicamente se especifique qué librería y versión es utilizada por el proyecto, y Maven se encargará de descargarla y empaquetarla dentro de la misma aplicación.

- **Ejecución de Test:** Se puede especificar que Maven ejecute automáticamente los Tests Unitarios (normalmente JUNIT) de un paquete específico. En función del resultado de esos tests Maven puede actuar de una manera u otra, continuando con el ciclo de empaquetado o no según se haya configurado.

- **Empaquetado:** Las aplicaciones Java tienen una serie de empaquetados específicos según la naturaleza de la aplicación. Las librerías o programas de

escritorio suelen ir empaquetados en ficheros JAR, las aplicaciones web suelen ir en ficheros WAR, y Oracle Application Server suele gestionar ficheros EAR. Cada tipo de empaquetado tiene una estructura determinada, y Maven ayuda a gestionarlo.

- **Despliegue:** Una vez empaquetado, el resultado puede tener que ir a sitios diferentes. Una librería es posible que se quiera dejar en un repositorio de Maven, y una aplicación Web quererla llevar directamente a un contenedor de aplicaciones. Maven permite la configuración de lugares donde enviar dicho paquete resultado, ya sea en la máquina local como en red.

- **Gestión de entornos:** Una misma aplicación puede tener como objetivo distintas instalaciones (máquina de desarrollo, servidores de Test, Pre-producción, Producción, etc…), para ello lo más probable es que utilicen recursos diferentes (conectores de bases de datos, configuraciones de logs, preferencias de la aplicación, etc…). Maven permite la gestión de diferentes recursos según perfiles o parámetros en la ejecución del empaquetado.

Las aplicaciones gestionadas por Maven, suelen seguir la siguiente estructura de carpetas:

- Carpeta del proyecto:
  - **/src/main/java:** Contiene los ficheros fuentes (JAVA)
  - **/src/main/resources:** Contiene los ficheros de recursos o configuración.
  - **/src/test/java:** Contiene los ficheros de test, normalmente JUNIT.
  - **/src/test/resources:** Contiene ficheros de recursos utilizados por los tests como pueden ser ficheros de configuración de los mismos, o de entrada de datos…
  - **/pom.xml:** Fichero de configuración de Maven.

2.1.4 Hibernate

Este es el núcleo del desarrollo del proyecto. Es un framework de persistencia para Java. Es un proyecto iniciado por una serie de programadores que posteriormente fueron contratados por la empresa JBoss para darle un soporte al proyecto.

Permite de manera sencilla abstraer al desarrollador en la tarea de trasladar un modelo relacional de Base de Datos al modelo de clases de dominio o negocio de Java. Este tipo de herramientas se les conoce como ORM (Object-Relational Mapping , mapeo objeto-relacional). Lo que hacen es mediante algún tipo de configuración automatizar la transformación de los datos, de manera que el desarrollador, una vez hecha dicha configuración (también comentar que hay herramientas que realizan dicha configuración a partir de una base de datos existente) se despreocupa que en el sistema existe una Base de Datos y trabaja únicamente con objetos Java.

Dicha configuración se puede realizar de dos maneras:

- **Ficheros de configuración:** Son ficheros HBM.XML. Existe uno por entidad donde se especifica el nombre de la tabla, el nombre de la Clase, y la relación de Columna-Campo de clase.

- **Anotaciones:** Utilizadas dentro del código Java. Este método se verá más adelante en la memoria, pues es el que se ha utilizado en el desarrollo del proyecto. La decisión viene tomada por las siguientes razones: se ahorran del proyecto una cantidad considerable de ficheros de configuración, se eliminan declaraciones de campos duplicadas (al tener que declararlos en la Clase y en el fichero), reducir el número de puntos en la aplicación donde realizar cambios frente a modificaciones en el modelo de datos.

Hibernate ofrece una serie de alternativas para realizar **consultas** de datos de las cuales las más importantes son las siguientes:

- **HQL:** Es un lenguaje parecido al SQL pero más cercano a la nomenclatura de lenguaje orientado a objetos. Normalmente se construye un String que contiene la sentencia y se lanza con el motor que ofrece Hibernate, obteniendo como resultado uno objeto o una lista de ellos.

- **Criteria:** Se construye la consulta utilizando una serie de objetos Java facilitados por Hibernate haciendo referencia directa a las propiedades de las clases de negocio. En este caso las únicas cadenas que hay que construir es cuando se referencia una propiedad de un Objeto que se suele realizar por nombre.

2.1.5 Spring

Es un framework inicialmente desarrollado para Java que actualmente presenta también versión para .NET. En principio surgió como alternativa más liviana de la metodología de programación que imponían los EJB (Enterprise Java Beans).

Actualmente el proyecto Spring está dividido en muchas capas, como configuración, web, persistencia, inversión de control, tests, etc...

En el proyecto se utiliza una parte reducida de todo el framework, la que hace referencia a la inyección de dependencias.

Al igual que Hibernate, ofrece dos maneras de configuración, una por ficheros XML, y otra por anotaciones. En este caso se utiliza una manera mixta que consiste en tener un fichero XML de configuración general, donde se inicializan y configuran algunos otros frameworks (como Log4j e Hibernate), y anotaciones para realizar la inyección de dependencias.

Tal y como se usa en el proyecto, podemos hacer una abstracción de Spring, y verlo como un contenedor de objetos, donde se inicializan, se registran, y se asocian unos con otros. En contraposición con la manera normal de instanciar un objeto, donde se utiliza la sentencia **new** y o bien en dicha sentencia como parámetros o bien utilizando posteriormente los **setters** de dicha instancia se le asocian las clases de las que dependa (por ejemplo, un servicio suele depender de uno o varios *DAOs* (Data Access Object) u objetos de acceso a datos), en este caso se especifica por configuración una dependencia entre clases y se delega a dicho contenedor la tarea de instanciarlos y asociar unas instancias con otras. La inyección de dependencias está relacionada con el patrón de diseño de inversión de control, de manera que el flujo normal de ejecución de una aplicación se invierte delegando el control en una entidad superior en función de unos parámetros y no es la propia entidad programada la que toma decisiones (en este caso la decisión de asociar la instancia dependiente).

2.1.6 JUNIT

Es un framework para realizar pruebas unitarias automatizadas de la aplicación. Permite tener una batería de pruebas pequeñas con las cuales detectar si cualquier cambio en el código realiza algún cambio en los resultados esperados y así detectar errores en el mantenimiento de aplicaciones.

Las pruebas unitarias tienen las siguientes características:

- **Automatizable:** No requieren acciones manuales, todo dato que deba ser facilitado a la aplicación deberá ser por código o por recursos externos siempre disponibles.

- **Completas:** Deben cubrir la mayor cantidad de código. Existen herramientas capaces de calcular el porcentaje de código cubierto por pruebas.

- **Repetibles:** Deben poder ser ejecutadas todas las veces que hagan falta, esto suele hacer referencia a realizar cambios en algún contexto, como pueda ser insertar registros en una base de datos.

- **Independientes:** La ejecución de una prueba no debe afectar al de otra. Si por ejemplo en una prueba se inserta un registro, no se debería realizar otra prueba para probar a leerlo, si no que debería ser la propia segunda prueba la que ella misma insertara otro registro para leerlo.

JUnit en concreto, lo que aporta es una serie de sentencias para evaluar los resultados llevados a cabo por una ejecución y compararlos con los resultados previstos que se deberían obtener. Por lo general los entornos de programación en Java ofrecen buenas herramientas para la ejecución automática de baterías de dichas pruebas y la detección mediante alertas de posibles errores.

A comentar que si bien las pruebas unitarias, si se realizan correctamente, pueden ayudar a detectar muy tempranamente un error al cambiar código existente, hay que

tener cuidado en la confianza, pues una prueba mal hecha puede generar una falsa confianza en un error no detectado.

2.1.7 Log4j

Desarrollado por Apache Software Foundation, facilita las tareas de desarrollo con *logs* (mensajes de registro). Los logs son muy importante en toda aplicación, pues pueden ser indicadores utilizados para diferentes aspectos de supervisión como detección de errores, identificación de los mismos, estudio del rendimiento, estudio de la traza de datos, etc...

Este framework ayuda al desarrollo de la aplicación al permitir una configuración en tiempo de ejecución de distintos aspectos de dichos logs:

- **Nivel:** Dada la naturaleza del mensaje éste puede ser de aviso, de debug, de información, de error... De esta manera se puede configurar cada mensaje cómo tiene que ser mostrado, y filtrarlos en función del momento o entorno.

- **Salida:** Hay muchas maneras de enviar un mensaje, ya sea a un fichero, por consola, o incluso enviarlo por sockets a un servidor remoto. En el caso de los ficheros se puede configurar también una política de retención, como borrar cada X días, o reutilizarlos de manera circular, etc...

- **Formato:** Se puede configurar tanto qué información se quiere mostrar (fecha, hora, clase afectada, método afectado, línea afectada...) como también el formado de dichos datos(fecha, hora...)

- **Clases:** Para cada clase o paquete de la aplicación podemos definir si se desea a qué nivel de log y en qué salida se quiere configurar.

2.2 Webservices

El proyecto se basa en una arquitectura *SOA* (Service Oriented Architecture, o Arquitectura Orientada a Servicios). Básicamente define una serie de servicios bien definidos, a los que se pueden acceder mediante un protocolo estándar de comunicaciones. En el proyecto se utilizan en concreto Web Services (Servicios Web) que funcionan sobre el protocolo *HTTP* (Hyper Transfer Protocol), el cual es usado por el *SOAP* (Simple Object Access Protocol) que define el intercambio de información entre dos puntos utilizando ficheros XML como base.

Los Servicios Web definen su interfaz mediante ficheros *WSDL* (Web Services Description Language). Estos ficheros tienen el formato de ficheros XML, donde básicamente se describe los tipos de datos utilizados, los métodos ofrecidos (también llamados mensajes) con sus parámetros y resultado.

Esto permite que dichos servicios puedan ser invocados por diferentes entidades, ya no solo que estén fuera del propio proyecto, si no que estén escritas en cualquier lenguaje de programación, únicamente conociendo el punto de acceso y la interfaz definida.

2.2.1 JAX-WS

Para el desarrollo de dichos Servicios se ha utilizado JAX-WS (Java API for XML Web Services), un API dentro de las librerías estándares de Java EE que se desarrolló en su versión 5 por Sun Microsystems. Permite, mediante anotaciones que más adelante se explicarán, abstraer al desarrollador de toda la parte de utilización del protocolo de comunicación y transformación de datos en XML, únicamente dedicándose a la programación de la lógica del propio negocio.

2.2.2 SOAP-UI

Es una herramienta autónoma de testeo de arquitecturas SOA como la nuestra. Permite de manera sencilla probar Web Services sin tener que desarrollar a mano un cliente que haga uso de ellos.

La aplicación usa como unidad de funcionalidad los proyectos. Cada proyecto puede albergar varios servicios, creados automáticamente especificando la dirección de algún fichero WSDL. Cada servicio contiene una referencia a cada uno de los métodos que lo componen, generada también automáticamente al crearse él mismo. Estas referencias a métodos pueden contener una o varias peticiones, las cuales se definen como un XML con el formato especificado por el servicio a las cuales les puedes modificar los nodos que hacen referencia a los parámetros de entrada. Estas peticiones son las que pueden ser lanzadas para probar el servicio, dando como resultado un XML definido también por dicho método. Esto permite de una manera muy sencilla probar tanto la disponibilidad del servicio, como el buen funcionamiento del mismo, como incluso averiguar el funcionamiento de algún servicio publicado del que desconocemos el resultado esperado ante algunos parámetros de entrada específicos (incluso la detección de errores en los propios parámetros de entrada).

3. Desarrollo del proyecto

3.1 Aplicación

Como se ha introducido, la finalidad principal del proyecto es el desarrollo de una aplicación gestora de un catálogo de máquinas virtuales, *VMRC* (Virtual Machine image Repository & Catalog).

Este proyecto se limita al desarrollo únicamente de una parte de dicha aplicación, en concreto la que realiza las tareas de servidor. Pero es interesante dar una rápida visión general a la arquitectura completa de la solución.

3.1.1 Introducción a la aplicación

VMRC es un proyecto del Grupo de Grid y Computación de Altas Prestaciones (GRyCAP), en el Instituto de Instrumentación para Imagen Molecular (I3M) en la Universidad Politécnica de Valencia (UPV). Ofrece servicios de un catalogo de imágenes de máquinas virtuales (denominadas Virtual Machine Images, VMI) almacenadas en su propio repositorio y en catálogos externos.

Entre los principales servicios que se propone ofrecer se encuentran los siguientes:

- **Búsqueda de VMIs**: Ofrecer un motor de búsqueda de imágenes mediante sus especificaciones tanto tipo de hypervisor, como software y hardware. Para esto se ha utilizado un lenguaje propio, basado en una colección registros clave-valor específicos para representar las diferentes propiedades del sistema y una serie de operaciones para evaluar la búsqueda. Una de las características del lenguaje que lo hacen más potente es la capacidad de realizar búsquedas aproximadas, permitiendo asignar una puntuación a cada característica encontrada.

- **Administración de VMIs:** Los usuarios podrán guardar sus imágenes especificando sus parámetros comentados en el punto anterior, así como su eliminación o edición, para su posterior consulta o descarga. Este servicio hace uso de los protocolos HTTP y FTP para la transferencia de los ficheros requeridos.

3.1.2 Arquitectura

La solución completa se compone de las siguientes partes:

- **Servidor:** Se encarga de la lógica y la gestión de los recursos necesarios para llevar a cabo las tareas de los servicios previamente explicados. Estos recursos

son tanto la base de datos donde se almacena la información de las VMIs, como el repositorio donde se guardan físicamente dichas imágenes, y los mecanismos para el acceso a catálogos de terceros. Será desplegado en un servidor que admita las tecnologías empleadas en su desarrollo, en principio un contenedor de aplicaciones Apache Tomcat, aunque no está limitado a ello.

- **Cliente:** Son librerías desarrolladas también en Java, con la JDK 1.6, que gestionan las llamadas a dichos servicios. Los clientes están en principio pensados para ser de dos tipos:

    - **Comandos de consola:** Para poder acceder a los servicios mediante un terminal o Shell, con total ausencia de interfaz gráfica. Esta es la implementación más directa desde dichas librerías al no requerir una interacción elaborada con el usuario.

    - **Aplicación Web:** Desarrollada en Java para aprovechar las librerías de acceso al servidor, aunque se recuerda que gracias a que los servicios son Web Services, ésta podría haber estado programada en cualquier lenguaje que lo permitiera, desde una simple aplicación HTML con Javascript, hasta una aplicación desarrollada en tecnologías similares a Java como .NET.

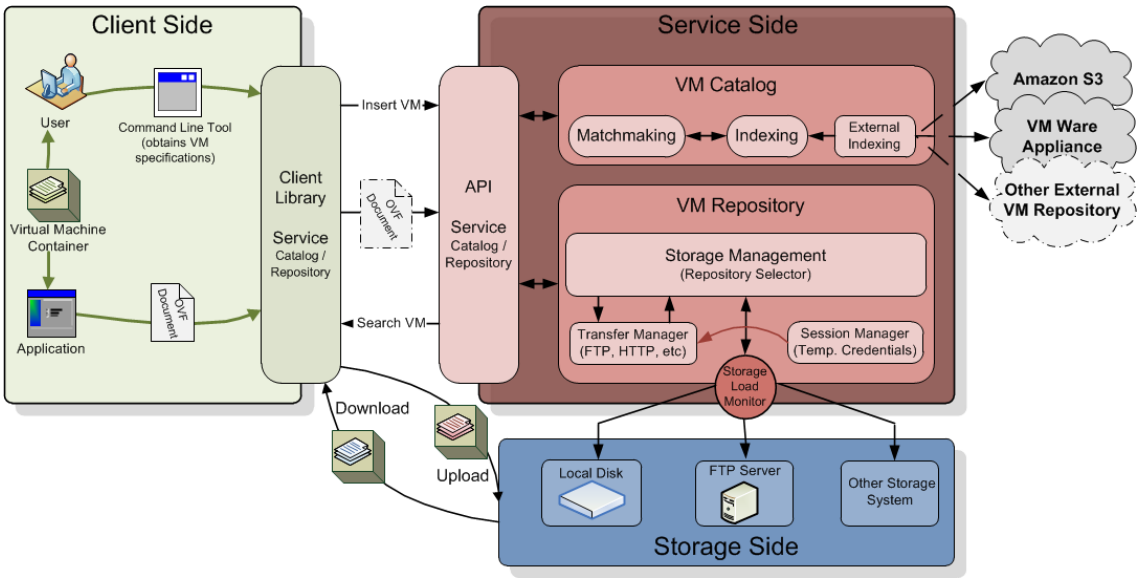

Figura 1. Arquitectura del servicio de repositorio y catalogación de imagines de máquinas virtuales (VMRC).

3.1.3 Modelo de datos

El modelo de datos de la aplicación representa en forma relacional la información que se puedan guardar referentes a cada VMI almacenada, así como de sus usuarios.

Esta representación intentará ser lo más fiel posible a las clases que se utilicen en la aplicación, que desarrollada con el paradigma de la Programación Orientada a Objetos supone la mejor aproximación a la realidad.

En este momento podemos hablar de las siguientes entidades abstractas de las cuales veremos su representación en ambos formatos:

- **Imagen de Máquina Virtual:** Describe una VMI tal cual se almacena en el sistema. Es la entidad que más información almacena, siendo esta sus especificaciones de tipo, software, hardware, así como información del usuario de inicio de sesión al ser utilizada. Tanto en la aplicación como en la base de datos está representada por la entidad **VMI**.

- **Aplicación:** Describe una aplicación que pueda ser instalada en una VMI. Contiene información sobre su nombre, ruta de instalación y versión. Tanto en base de datos como en la aplicación viene representada por la entidad **Application**. Una VMI puede contener varias aplicaciones. En base de datos, dicha relación está representada por la tabla VMI_APP.

- **Sistema Operativo:** Describe un sistema operativo que pueda ser instalado en una VMI. Contiene información sobre su nombre, variante o distribución, y versión. Tanto en la aplicación como en la base de datos está representado por la entidad **SO**. Una VMI tiene únicamente un Sistema Operativo, y su relación está definida en la propia entidad VMI.

- **Usuario:** Describe un usuario del sistema (no confundir con el de inicio de sesión de las VMI). Contiene únicamente información sobre el nombre, el password y sus permisos (explicado a continuación). Es el que se usará para gestionar los permisos de acceso a la aplicación así como a cada una de las imágenes en el repositorio o los datos de estas. Tanto en la aplicación como en la base de datos viene representado por la entidad **User**.

- **Listas de Control de Acceso:** Describen las diferentes operaciones definidas en el sistema. Existen de dos tipos:

    - De usuario: Sirve para granular la seguridad en función del usuario. Define qué acciones no pueden ser realizadas por un usuario si su valor es **none.** Éstas se definen en la base de datos como una propiedad del Usuario.

- De VMI: Sirve para controlar quien puede llevar a cabo acciones sobre una VMI. Contiene la información sobre el usuario que puede realizar las acciones definidas sobre cada VMI, admitiendo como valores **all** o **owner** en función de si esa acción la puede realizar todos o únicamente el dueño de la imagen respectivamente.

Tanto en la aplicación como en la base de datos se representan por la entidad **ACL**.

3.2 Desarrollo

Ahora se explicará cómo se han desarrollado los diferentes aspectos de la aplicación:

3.2.1 Acceso a datos: Hibernate

Esta es una de las partes más importantes del proyecto. La idea era conseguir un mecanismo que simplificara el acceso a los datos, sustituir las anteriores sentencias SQL por algo que permitiera realizar consultas que de cara a cambios en el modelo fuera más sencillo de implementar.

Hibernate permite definir una equivalencia entre entidades relacionales de la base de datos (Tablas) y los objetos de dominio en Java (POJOs). Así, todas las acciones que realices con dichos objetos se verán reflejadas en la base de datos tal y como se ha especificado en dicha equivalencia, y al realizar cambios en la definición de los objetos, únicamente se deberá realizar cambios en esa configuración, siendo transparente para el resto de la aplicación.

Hay dos maneras de configurar dicha equivalencia. Una es escribiendo un fichero XML por cada entidad mapeada. En dicho fichero se especifica cada una de las propiedades del objeto y se define a qué columna hace referencia en la base de datos, así como las relaciones con otros objetos u otras propiedades especiales como la generación del identificador de la tabla. Otra opción es utilizando anotaciones del estándar JPA (Java Persistence API), en el propio POJO, que es la opción que se ha utilizado en este proyecto debido al ahorro de líneas de código y fichero que ello implica.

Por ejemplo, la clase VMI quedaría así:

```
@Entity
@Table(name = "VMI")
public class VMI implements Comparable<VMI> {

 @Id @GeneratedValue @Column(name = "VMI_ID")
 private Long id;

 @NaturalId @Column(name = "VMI_NAME", length = 255)
 private String name;

 private Date timestamp;
 private String hypervisor;
 private String arch;
 private String diskSize;
 private String location;
 private String userLogin;
 private String userPassword;
 private String checksum;
 private String owner;

 @ManyToOne (cascade=CascadeType.ALL) @JoinColumn(name="ACLID")
 private ACL acl;

 @OneToOne @JoinColumn(name = "OS_ID")
 private OS os;

 @ManyToMany(fetch=FetchType.EAGER)
 @JoinTable(name = "VMI_APP",
       joinColumns = { @JoinColumn(name = "VMI_ID") },
       inverseJoinColumns = { @JoinColumn(name = "APP_ID", insertable = true) })
 private Set<Application> applications = new HashSet<Application>();

 @Transient
 private int rankValue;
```

Explicación de cada uno de los elementos:

Anotaciones de Clase:

@Entity: Esta anotación indica que esta clase está persistida en base de datos y habrá que analizar el resto de anotaciones para configurar la equivalencia.

@Table: Indica la tabla de la base de datos que será la equivalente a esta clase del dominio. Su nombre viene dado por el atributo **name** de esta anotación. En este caso **VMI**.

Anotaciones de Campo:

@Id: Indica que el siguiente atributo será el ID de la tabla.

@Generated: Indica que el siguiente atributo será generado automáticamente. Esto es que el usuario no puede darle un valor especifico, si no que se generará al momento de guardar el registro en la base de datos.

@Column: Indica la tabla de la base de datos a la que hará referencia el siguiente atributo. Esta anotación tiene varias propiedades para definir restricciones sobre el tipo de datos, como longitud, formato, etc... En este caso el atributo **Id** de la clase hace referencia a la columna **VMI_ID** de la base de datos.

@NaturalId: Indica que el siguiente atributo no será el propio ID de esa tabla, si no un atributo que identifica unívocamente a los registros de forma semántica.

@ManyToOne: Indica que el siguiente atributo tendrá una relación con otra entidad. Se utiliza junto con otra anotación para especificar la columna de la base de datos que hace de clave ajena (**@JoinColumn**). En este caso la propiedad **acl**, de la entidad vendrá definida por el ID que contenga la columna **ACLID**. Una vez definida así, automáticamente la propidad **acl** será rellenada con la entidad **ACL** que corresponda, sin que el desarrollador se tenga que preocupar por ello.

@OneToOne: El comportamiento es idéntico a **ManyToOne** con la característica que solo podrán haber una referencia a cada entidad destino y viceversa.

@ManyToMany: Esta es la relación más complicada de todas (muchos a muchos), pues necesita una tabla auxiliar para llevarse a cabo. Dicha tabla se especifica mediante la anotación **@JoinTable** y se especifican también que columnas de dicha tabla hacen referencia a las dos entidades referenciadas.

@Transient: Indica que el siguiente campo será temporal y no se deberá persistir en la base de datos.

Sin anotación: Por defecto, todas los elementos de un POJO serán tratados como si tuvieran la anotación **@Column(name="<nombre>")**. Es decir, estarán relacionados con la columna homónima de la base de datos.

3.2.2 Inyección de dependencias: Spring

Para orquestar todos los fragmentos de la aplicación se ha utilizado el mecanismo de inyección de dependencias o inversión de control, para lo que ha sido de gran ayuda el framework Spring. Esta inyección de dependencias realmente consiste en la inicialización de la gran mayoría de componentes (instanciación de clases necesarias y configuración de las mismas), y además la asociación de dichas clases. Por poner un ejemplo de esto mismo:

En esta aplicación, los servicios se apoyan de los DAO, que a su vez se apoyan en Hibernate, que necesita un *SessionFactory* para trabajar.

1. Spring creará una instancia de cada uno de los componentes.
2. Llamará a los métodos pertinentes para configurar **Hibernate** y asociarle dicha sesión.
3. Llamará a los correspondientes **setters** de dichas capas para asociar la sesión de Hibernate al **DAO**, y el DAO al **Servicio**.

Todo parte de un XML inicial de configuración, **applicationContext.xml**. Se explicará cada una de sus acciones:

```
<context:component-scan base-package="org.grycap.vmrc" />
<context:annotation-config/>
   <tx:annotation-driven transaction-manager="transactionManager" proxy-
target-class="true"/>
```

Este fragmento sirve para especificar que la configuración de los componentes y el control de las transacciones va a ser mediante anotaciones en los Beans, en concreto en el paquete **org.grycap.vmrc**.

```
<bean                                                          id="log4jInitialization"
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
       <property name="targetClass" value="org.springframework.util.Log4jConfigurer" />
       <property name="targetMethod" value="initLogging" />
       <property name="arguments">
              <list>
                     <value>classpath:log4j.properties</value>
              </list>
       </property>
</bean>
```

Esta es la inicialización del sistema de logs de la aplicación.

```
<!-- Se inyecta la sesión en el contexto -->
<bean id="sessionFactory" class="org.springframework.orm.hb3.annotation.SessionFactoryBean">
        <property name="configLocation" value="classpath:hibernate.cfg.xml" />
        <property name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration"/>
        <property name="packagesToScan" value="org.grycap.vmrc.entity"/>
</bean>

<!-- Hibernate Template -->
<bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
        <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Aquí se inicializa Hibernate, indicándole que el fichero de configuración es **hibernate.cgf.xml** incluido en el classpath, que la configuración del mapeo entre clases y base de datos será por anotaciones y que se encuentran en el paquete **org.grycap.vmrc.entity**.

Por parte de los Beans, la configuración se realizará mediante las siguientes anotaciones.

@Repository:
Anotación de **Clase**.
Se aplica a los DAOs para indicar que es una fuente de datos.

@Service:
Anotación de **Clase**.
Se aplica a los servicios. Realmente es como la anterior, pero tiene una diferencia simplemente semántica.

@Autowired:
Anotación de **Campo**.
Se aplica a los campos de las clases que quieres que sean autoinyectados. Esto es, cuando se instancie dicha clase mediante Spring, buscará entre sus clases instanciadas y la que coincida será la que use de parámetro al realizar la llamada al setter correspondiente.

3.3 Base de Datos

Gracias a tecnologías ORM como en este proyecto Hibernate, se ha conseguido una alta independencia del Sistema Gestor de Bases de Datos utilizado, únicamente teniendo en cuenta que debe ser de tipo relacional compatible con el estándar SQL.

Aun así veo interesante comentar las tecnologías empleadas durante el desarrollo del proyecto. En este caso se ha trabajado con dos sistemas de filosofía bastante diferente.

El proyecto se comenzó utilizando HSQLDB (evolución de HypersonicSQL). Es un sistema de bases de datos en memoria. Para la fase de desarrollo tiene una serie de ventajas como su ligereza, que no requiere instalación o arranque extras. En este proyecto se dejó de utilizar porque nos presentó problemas a la hora de consultar el estado de la base de datos al mismo tiempo que la aplicación estaba en ejecución con conexiones abiertas.

Al final se optó por utilizar MySQL, ya que es uno de los sistemas que mejor razón requisitos de máquina/potencia ofrece, así como su alto grado de uso. Se pueden conseguir herramientas como HeidiSQL que son interfaces para gestionar el sistema que son muy sencillas y completas a la vez, con eso nos bastaba para poder desarrollar y testear completamente la aplicación.

3.4 Web Services

Como se ha explicado, la parte servidora de todo el producto ofrecía una interfaz SOAP, para dar más libertad a las aplicaciones clientes en cuanto al lenguaje utilizado al usar un estándar que funciona sobre el protocolo HTTP.

Para ello se utilizó las librerías nativas de Java, en concreto la tecnología JAX-WS que facilita el desarrollo al automatizar toda la parte que concierne a la comunicación HTTP o parseo de los mensajes en XML. Transforma directamente tus clases de servicios con unas simples anotaciones que ahora se comentarán:

@WebService: Anotación de Clase. Indica que la clase anotada es un Webservice y la procesará para ofrecer los métodos indicados como llamadas SOAP.

@SOAPBinding(style = Style.DOCUMENT): Las llamadas serán transformadas a documentos XML como define el protocolo.

@WebMethod: Anotación de Método. Indica que el método anotado será ofrecido por la interfaz de webservice.

@WebParam: Anotación de Parámetro. Indica que el parámetro de la función es un parámetro de entrada del servicio y deberá ser parseado en XML.

Con esto ya solo queda configurar la aplicación mediante el fichero de configuración **web.xml**, añadiendo una entrada para el **servlet** que escuchará las peticiones al webservice. Será necesario con añadir lo siguiente:

```
<listener>
    <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
</listener>

 <servlet>
    <servlet-name>vmrc</servlet-name>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>vmrc</servlet-name>
    <url-pattern>/vmrc</url-pattern>
      </servlet-mapping>
```

3.5 Api de los servicios ofrecidos

Ahora se comentará qué métodos se han implementado y publicado como servicios que ofrecerá la aplicación para que sean consumidos por los clientes.

public VMI addVMIByObject(@WebParam VMI vmi) **throws** Exception

Registra una imagen máquina virtual en el catálogo.

Parámetros:

- VMI vmi: Objeto VMI que describe una imagen de máquina virtual. Puede ser serializado en XML para su transportación.

Excepciones:

- Error de conexión a la base de datos.
- Insuficientes permisos del usuario que intenta realizar el registro.

public VMI addVMI(@WebParam String vmiDescStr) **throws** Exception

Registra una imagen de máquina virtual en el catálogo.

Parámetros:

- String vmiDescStr: Cadena que describe la imagen en un formato determinado.

Excepciones:

- Error de conexión a la base de datos.
- Error en el formato de la descripción de la imagen de máquina virtual.
- Error en los permisos del usuario que intenta realizar el registro.

public List<VMI> search(@WebParam String vmiDescStr) **throws** Exception

Realiza una búsqueda de imágenes de máquinas virtuales con un lenguaje específico.

Parámetros:

- String vmiDescStr: Cadena que describe la búsqueda en lenguaje **vmidl**

Excepciones:

- Error de conexión con la base de datos.
- Error en el formato de la cadena de búsqueda.

public List<VMI> list() **throws** Exception

Realiza un listado de imágenes de máquinas virtuales a las cuales se tenga permiso de lectura.

Parámetros: No recibe ningún parámetro.

Excepciones:

- Error de conexión con la base de datos.

public void delete(@WebParam String vmiName) **throws** Exception

Elimina del catálogo la imagen de máquina virtual pasada como parámetro.

Parámetros:

- String vmiName: Nombre de la imagen de máquina virtual a eliminar.

Excepciones:

- Error de conexión a la base de datos.
- El nombre proporcionado no existe.
- Insuficientes permisos del usuario que intenta realizar el borrado.

public void addUser(@WebParam String userName, @WebParam String userPassword) **throws** Exception

Añade un usuario al sistema.

Parámetros:

- String userName: Nombre del usuario.
- String userPassword: Contraseña del usuario.

Excepciones:

- Error de conexión a la base de datos.
- Insuficientes permisos del usuario que intenta añadirlo.
- Nombre de usuario ya existente.

public void deleteUser(@WebParam String userName) **throws** Exception

Elimina un usuario registrado en el sistema.

Parámetros:

- String userName: Nombre del usuario.

Excepciones:

- Error de conexión a la base de datos.
- Privilegios insuficientes del usuario que desea borrar.
- Usuario no existente.

public String getOVFByVMI(@WebParam String vmiName) **throws** Exception

Obtiene la descripción de una imagen de máquina virtual. Dicha descripción viene definida en lenguaje vmidl.

Parámetros:

- String vmiName: Nombre identificativo de la imagen de máquina virtual.

Excepciones:

- Error de conexión a la base de datos.
- Insuficientes privilegios del usuario.

public void changeVMIAcl(@WebParam String vmiName, @WebParam String operation, @WebParam String perm) **throws** Exception

Modifica las listas de acceso de una imagen de máquina virtual.

Parámetros:

- String vmiName: Nombre identificativo de la imagen de máquina virtual.
- String operation: Operación sobre la imagen de máquina virtual que se desea modificar.
- String perm: Valor que se desea asignar a esa operación.

Excepciones:

- Error de conexión a la base de datos.
- Imagen de máquina virtual no existente.
- Operación no valida.
- Permiso no valido.

public void changeUserAcl(@WebParam String userName, @WebParam String operation, @WebParam String perm) **throws** Exception

Modifica las listas de acceso de un usuario.

Parámetros:

- String userName: Nombre identificativo del usuario.
- String operation: Operación sobre la imagen de máquina virtual que se desea modificar.
- String perm: Valor que se desea asignar a esa operación.

Excepciones:

- Error de conexión a la base de datos.
- Usuario no existente.
- Operación no valida.
- Permiso no valido.

public FTPTransferParams requestToUploadVMI(@WebParam String vmiName, @WebParam String vmiFileName) **throws** Exception

Realiza una petición para subir un fichero al catálogo determinando a qué registro de máquina virtual va asociado, y la ruta del fichero que contiene dicha imagen de máquina virtual.

Parámetros:

- String vmiName: Nombre identificativo de la máquina virtual registrada.
- String vmiFileName: Ruta del fichero de máquina virtual que se desea subir.

Excepciones:

- Error en la conexión a base de datos.
- Máquina virtual no existente.
- Fichero no encontrado.
- Privilegios insuficientes del usuario que realiza la petición.

public FTPTransferParams requestToDownloadVMI(@WebParam String vmiName) **throws** Exception

Realiza una petición para descargarse un fichero de imagen de máquina virtual. Devuelve una descripción de FTP de donde poder realizar dicha descarga.

Parámetros:

- String vmiName: Nombre identificativo de la imagen de máquina virtual.

Excepciones:

- Error de conexión a la base de datos.
- Máquina virtual no existente.
- Privilegios insuficientes del usuario que realiza la petición.
- Error de conexión con el servidor FTP.
-

3.6 Conexión con el Cliente y la Interfaz Web

La funcionalidad del Cliente por consola, como el de la interfaz Web no son objeto de este proyecto, pero sí en cambio sería interesante comentar el mecanismo por el cual se deberían conectar al servidor desarrollado.

Lo primero a comentar, como se ha dicho anteriormente, este proyecto ofrece una capa de servicios web, lo que nos abstrae de la tecnología utilizada en ambos lados de esa comunicación. Así que dichos clientes no deberían de tener porque estar escritos con el lenguaje Java.

La funcionalidad de los mismos también es indiferente en este estudio, pero es fácil de hacerse una idea, que deberían servir como interfaces de cara al usuario para manejar todos los servicios que hemos desarrollado. Así pues dichos clientes deberían tener una capa de comunicación de Web Services que encapsulara las llamadas a cada uno de los métodos del servidor al igual que en este proyecto se han encapsulado las llamadas a las operaciones en la base de datos. Por último quedaría utilizar dicha encapsulación en los métodos de la interfaz.

3.7 Disponibilidad del Servicio

Los desarrollos realizados en este PFC han sido integrados dentro de la arquitectura de VMRC y han sido liberados bajo licencia Apache. Se encuentran disponibles en http://www.grycap.upv.es/vmrc

4. Conclusiones

Este proyecto pretendía el desarrollo de una aplicación que diera un servicio que hoy en día, pese al auge de la virtualización no hay suficientes soluciones disponibles. Pese a que los requisitos de implementación tenían acotadas ciertas tecnologías a utilizar, como el lenguaje de programación Java, la librería Hibernate, el repositorio Subversion, y la interfaz de Web Services, quedaba a su vez mucha libertad para utilizar cualquier otra tecnología que estuviera al alcance, cosa que lo ha hecho verdaderamente atractivo.

Además de eso, debido a la naturaleza con la que se inició el proyecto, (esto es la modificación de un desarrollo existente debido a la dificultad de continuarlo) se pretendía establecer una metodología de programación que utilizando diferentes mecanismos que no trae por defecto un lenguaje en particular ayudara a crear aplicaciones que fueran fácilmente mantenibles, tanto de cara a arreglo de errores (se suele conocer como la fase correctiva) como a la inclusión de funcionalidad nueva (fase evolutiva).

Personalmente creo que se ha conseguido el objetivo, creándose una aplicación que a parte de dar la funcionalidad requerida se ha establecido una nueva forma de trabajar más limpia y sostenible.

Este proyecto ha sido un reto desde el punto de vista que cambia mucho la perspectiva del desarrollo de una aplicación en un lenguaje determinado al desarrollo de ese mismo lenguaje utilizando las herramientas alternativas existentes en el mercado. Aquí ya no solo cuenta el uso del lenguaje (que se supone ya controlado al completo o casi) si no el conocimiento o la facultad para buscar información de dichas herramientas extras, así como la capacidad de evaluar si dicha herramienta es la ideal para el propósito de tu desarrollo; habrá algunos frameworks más potentes que otros, pero para según qué aplicaciones a lo mejor no te interesa utilizar el potente, si no te vale con otro más liviano que tenga la funcionalidad que se desee utilizar.

A nivel personal este proyecto me ha enriquecido bastante, pues se ha utilizado muchísima tecnología que en la carrera pese a haber estado 5 años en el mundo java jamás se nos había enseñado ni siquiera su existencia, y se aprende a valorar las facilidades que aporta al desarrollo de un producto. También ha servido para aprender, o más bien hacer hincapié en la importancia de un desarrollo pensado en el futuro, fácilmente adaptable.

5. Bibliografía

La razón de que la gran mayoría de referencias sean páginas web y no libros es que el proyecto está enfocado a una serie de tecnologías de programación con proyectos continuos que cuelgan su propia documentación en formato electrónico, a parte de ser oficiales son mucho más accesibles hoy en día.

[1] Java: http://www.java.com/es/download/whatis_java.jsp

[2] Framework: http://es.wikipedia.org/wiki/Framework

[3] Hibernate: http://www.hibernate.org/

[4] Spring: http://www.springsource.org/

[5] Maven: http://maven.apache.org/

[6] Subversion: http://subversion.tigris.org/

[7] Anotaciones: http://es.wikipedia.org/wiki/Anotación_Java

[8] Robert C. Martin, Clean Code A Handbook of Agile Software Craftsmanship