FACULTAD DE CIENCIAS

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

Desarrollo de un cliente web para la exploración y visualización de cubos OLAP

(Development of a web client for exploring and visualizing OLAP cubes)

Para acceder al Título de
INGENIERO EN INFORMÁTICA

Autor: Sergio Rábago Bolado
Julio 2012

FACULTAD DE CIENCIAS

INGENIERÍA INFORMÁTICA

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por:	Sergio Rábago Bolado
Director del PFC:	Daniel San Martín Segura
Título:	Desarrollo de un cliente web para la exploración y visualización de cubos OLAP
Title:	Development of a web client for exploring and visualizing OLAP cubes

Presentado a examen el día:

para acceder al Título de
INGENIERO EN INFORMÁTICA

Composición del Tribunal:

Presidente (Apellidos, Nombre): González Harbour, Michael
Secretario (Apellidos, Nombre): Martínez Fernández, María del Carmen
Vocal (Apellidos, Nombre): Menéndez de Llano Rozas, Rafael
Vocal (Apellidos, Nombre): Zorrilla Pantaleón, Marta Elena
Vocal (Apellidos, Nombre): Sanz Gil, Roberto

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: Vocal

Fdo.: Vocal

Fdo.: Vocal

Fdo.: El Director del PFC

Agradecimientos

Como suele ser natural en cualquier proyecto fin de carrera, no es un trabajo solitario. Durante el desarrollo del mismo, uno se va dando cuenta de la importancia de las personas más cercanas que están a su alrededor (padres, hermanos, amigos y compañeros de universidad) y lo necesario que fue tenerlos cerca.

En primer lugar quiero agradecer a Daniel San Martín, por darme la oportunidad de realizar este proyecto dentro de la empresa Predictia, por guiarme durante la realización del mismo y ayudarme a solventar todos los problemas que aparecieron, así como sus propuestas de mejora, las cuales ayudaron en gran medida a crear un software competitivo y manejable.

También quiero agradecer la ayuda y el apoyo del resto de trabajadores y becarios de la empresa.

Por último, quiero agradecer a mis padres Marián y Jose Luis, mi hermana Lucía no sólo por comprenderme, sino aguantarme con la paciencia y cariño que demostraron.

Con este proyecto finaliza una etapa de mi vida, sin duda alguna una de las mejores y más enriquecedoras. Por todo ello y a todos, muchas gracias.

Sergio Rábago.

Resumen

Las tecnologías Business Intelligence permiten mediante el análisis y tratamiento de los datos extraer información orientada a la toma de decisiones. En la actualidad son muy utilizadas en el mundo empresarial para tratar y analizar conjuntos masivos de datos con objeto de obtener conclusiones y futuras estrategias que aporten una ventaja competitiva a la empresa en el sector donde compite.

Este proyecto ha sido realizado dentro de ese ámbito y a petición de la empresa Predictia Intelligence Solutions como solución a una necesidad. El objetivo ha consistido en desarrollar un componente reutilizable para el framework de creación de aplicaciones web Vaadin, que sirva para la visualización y navegación de cubos OLAP(On-Line Analytical Processing), estructuras de datos que por su configuración permiten el análisis de datos. Los requisitos exigidos fueron, desde el punto de vista del programador además de su correcto funcionamiento, sencillez de uso y facilidad para incorporarse en un nuevo proyecto software; y del usuario, que presentara una interfaz clara, sencilla y que permitiera las operaciones roll-up, drill-down, slice-dice, etc.

La conexión de la aplicación con los cubos está soportada gracias al servidor de cubos Mondrian, que se encargará de construir y mantener el cubo OLAP cuyos datos estarán almacenados en un repositorio relacional.

Palabras Clave

OLAP, Vaadin, Business Intelligence, Mondrian, Visor de cubos OLAP

Preface

Business Intelligence allows us to use the analysis and processing of data, extract information. Currently it is profusely used in the business area in order to analyze massive datasets, draw conclusions and future strategies which sum a competitive advantage in the segment where the company competes.

This project has been developed within this area, at the request of the Predictia Intelligence Solutions company, as a future solution for them. The aim is to develop a reusable Vaadin component for exploring and visualize OLAP (On-Line Analytical Processing) cubes, data structures that allow us data analazing. The requirements were, from the programmer standpoint, in addition to well-functioning, it must be easy to use and easy to include it into a new software project; and from the user standpoint, it must have a clear and simple interface. It must also allow roll-up, drill-down, slice-dice, etc. OLAP operations.

The cube connection is supported by server cubes Mondrian, which is responsible for building an maintaining OLAP cube, whose data will be stored in a relational repository.

Keywords

OLAP, Vaadin, Business Intelligence, Mondrian, OLAP cubes Viewer

Índice general

Índice de figuras

Capítulo 1

Introducción y contexto

En este capítulo se pretende poner en contexto la aplicación y describir el área de negocio en el que se encuentra inmersa, así como algunas claves para entender la tecnología que lo ha hecho posible.

Índice

1.1. Antecedentes

Desde hace unos años la recolección y almacenamiento de datos relacionados con las empresas (clientes, operaciones, transacciones...) ha experimentado un gran desarrollo gracias a los avances tecnológicos del mundo software y hardware. En consecuencia, las empresas disponen de una gran cantidad de datos que si fueran adecuadamente seleccionados y tratados les podrían ser muy útiles para conocer mejor su negocio y tomar decisiones de manera más informada.

Desde los años 90, las empresas han estado implantando sistemas de gestión empaquetados o a medida, que han dado solución a las necesidades diarias del negocio. Estos sistemas almacenan y gestionan cantidades ingentes de datos. Gran parte de esos datos no reciben ningún tipo de tratamiento adicional, pasando a engrosar los sistemas de backup e históricos de la empresa. Sin embargo se puede ir un paso más allá y analizar dichos datos para que sean productivos para la empresa y se puedan tomar decisiones teniendo en cuenta esos hechos. Esto puede ayudarnos a optimizar procesos, reducir costes, anticiparse a la

competencia... Por tanto necesitamos tecnologías y herramientas que permitan el análisis y la extracción de información útil de una gran cantidad de datos.

Las herramientas y tecnologías necesarias para llevar a cabo esa extracción de conocimiento se han denominado o englobado dentro de los **Sistemas de Soporte de Decisión (DSS)** o **tecnologías Business Intelligence (inteligencia de negocio)** que, en el fondo, permiten a los directivos de las empresas disponer de la información estratégica en línea y además, visualizarla y manejarla de forma gráfica y sencilla.

1.2. Bussiness Intelligence

El objetivo básico del Business Intelligence (Inteligencia de Negocio), es ayudar a mejorar la competitividad facilitando la información necesaria para la toma de decisiones. Mediante el uso de tecnologías y las metodologías de Business Intelligence pretendemos convertir datos en información y a partir de la información ser capaces de extraer conocimiento. Las soluciones BI incluyen la capacidad de acceder a la información desde varios medios, realizar consultas, informes y modelos de predicción.

Algunos ejemplo concretos de cómo el Bussines Intelligence nos puede generar beneficios, los encontramos en [4]:

- Beneficios tangibles:
  - Conocer mejor las características demográficas de nuestra zona de influencia.
  - Medir la efectividad de las campañas rápidamente.
  - Analizar la afinidad entre servicios o productos.
  - Analizar la productividad de los empleados.
  - Reducir o reasignar el personal necesario para llevar a cabo los procesos.
  - Aumentar el control de costes.
  - Disminuir gastos.
- Beneficios intangibles:
  - Mejorar el acceso a los datos a través de consultas, análisis o informes.
  - Mayor integración de la información.
- Beneficios estratégicos:
  - Toma de decisiones más rápida, informada y basada en hechos.
  - Mayor visibilidad de la gestión.
  - Mayor habilidad para analizar y dar soporte a estrategias.

La mayoría de organizaciones lleva un seguimiento de sus operaciones (servicios, ingresos, ventas...). Toda operación es susceptible de generar datos que pueden ser recogidos a través de sistemas transaccionales OLTP (Online Analytical Processing).

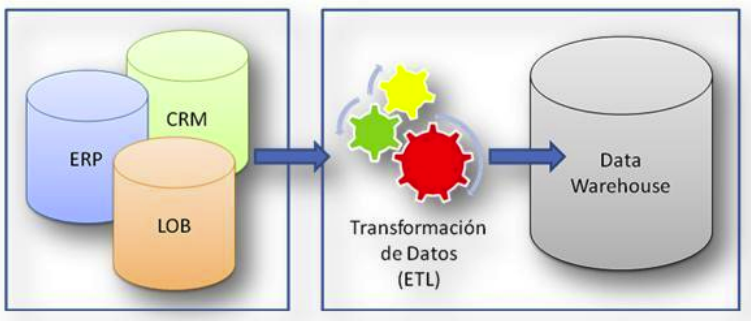
Los sistemas OLTP no sirven para proporcionar business intelligence por varios motivos:

- Son capaces de calcular agregados pero no están diseñados para hacer este tipo de consultas de forma masiva sobre volúmenes grandes de datos. El cálculo de medidas en el mismo OLTP disminuye su rendimiento y perjudica la gestión de las tareas diarias. Además de que el tiempo de respuesta a una petición de este tipo es demasiado alto.
- Puede no permitir el análisis histórico. Pueden generarse copias de back-up y no tener algunos datos accesibles on-line.
- Trabajar con OLTP requiere ciertos conocimientos técnicos: nombres de campos y tablas crípticos, complejas relaciones entre tablas (claves externas...). Con los sistemas OLAP también son necesarios algunos de estos conocimientos pero lo enmascaramos con la interfaz.

Por tanto, el primer paso para alcanzar una solución BI consiste en reunir toda la información útil de las diferentes fuentes disponibles (datos de la organización, redes sociales, datos demográficos, Internet...), almecenarla en una localización separada y organizar los datos de forma adecuada.

La migración y tratamiento de los datos es un proceso costoso y clave que determina en gran medida la calidad del resultado. Al extraer datos de varios sistemas OLTP surgen problemas en cuanto a formato, identificadores y calendarios empleados en cada una de las bases de datos de origen. Para solucionar este problema se realizan procesos ETL (Extracción, Transformación y Carga/Load).

Según Kimball[etl1] en el proceso de transformación de datos, y formateado de los mismos, para su almacenamiento dimensional, conlleva al menos el 70 % de la carga de trabajo del desarrollo de un data warehouse.



Una vez que tenemos toda la información reunida y organizada, podemos hacer uso de las herramientas de segmentación y análisis para extraer relaciones, tendencias, patrones... entre distintas variables de nuestro entorno.

1.2.1. Data Mart

Es un subconjunto de la información de un data warehouse, generalmente de un solo proceso de negocio, que se dirige a un determinado departamento/grupo de usuarios. Los data marts se pueden generar obteniendo datos de un data warehouse corporativo central o pueden ser creados independientemente de fuentes de datos independientes.

Podemos diseñar y construir un Data Mart como un modelo multidimensional, en el que las reglas de normalización se sustituyen por un método de diseño que gira alrededor de los "hechos". Los esquemas en estrella (Figura 1.1) y en copo de nieve (Figura 1.2) son la base del diseño de los Data Marts y permiten el acceso fácil y rápido a la información.

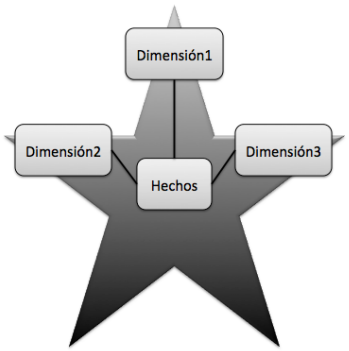


Figura 1.1: Diseño en estrella.

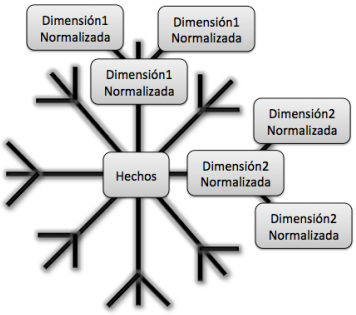


Figura 1.2: Diseño en copo de nieve.

Los datos que se utilizan en un Data Mart pueden ser clasificados en cuatro categorías: medidas, dimensiones, atributos y jerarquías.

- Medidas o métricas: Medición matemática de una variable de negocio. Qué quiero medir. Las tablas que contienen las medidas se llaman tablas de hechos.

- Dimensiones: Es una clasificación utilizada para expandir una medida agregada y ver las partes que la constituyen. Indica la perspectiva sobre la que quiero observar los datos.

- Atributos: son las columnas de la dimensión, elementos por las que se observará la medida.

- Jerarquías: En muchos casos una dimensión forma parte de una estructura más amplia con varios niveles. A esta estructura se llama jerarquía y podemos observarlo sobre todo en diseños en copo de nieve.

1.2.2. Tecnología OLAP

Cada vez que un usuario quiere ver un valor agregado el sistema debe calcularlo a partir de los datos del Data Mart, proceso en el que se invierte un tiempo considerable. Los sistemas OLAP, On-Line Analytical Processing, pretenden minimizar estos tiempos mientras

el usuario interactúa on-line con los valores. Al utilizar tecnología OLAP se pretende tener una serie de ventajas:

- Modelo de datos intuitivo y multidimensional que facilite la selección, recorrido y exploración de los datos.
- Lenguaje analítico de consulta que proporcione la capacidad de explorar complejas relaciones existentes entre los datos.
- Precálculo de los datos consultados con más frecuencia que permita mejorar la respuesta.

Los cubos son las estructuras de datos clave en OLAP, son subconjuntos de datos de un almacén de datos, organizados y sumarizados dentro de una estructura multidimensional. Los datos se sumarizan de acuerdo a los factores de negocio que se quieran analizar.

La definición del cubo es el primero de los tres pasos para su creación. Los otros pasos son, el especificar la estrategia de sumarización diseñando las agregaciones (elementos precalculados de datos), y la carga del cubo para procesarlo. En la definición del cubo se debe seleccionar la tabla de hechos y las medidas dentro de esa tabla. Después habrá que añadir las dimensiones que proporcionarán la descripción categórica al analizar las medidas.

Una vez diseñado y creado el cubo, mejorará considerablemente la respuesta del sistema, puesto que cuando un usuario quiera consultar una medida para un cierto conjunto de miembros dimensionales, el valor se leerá en una base de datos en lugar de calcularse en tiempo de consulta.

1.2.3. Arquitecturas OLAP

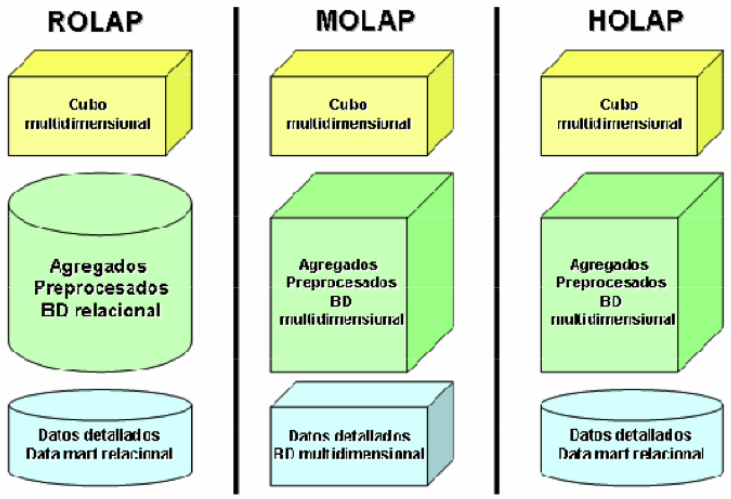


Figura 1.3: Arquitecturas OLAP

La arquitectura **ROLAP** (OLAP relacional) almacena las medidas detalladas en una base de datos relacional, que es la fuente del cubo. Es capaz de usar agregados precalculados si están disponibles, o de generar dinámicamente los resultados desde los datos originales.

Los usuarios finales ejecutan sus análisis multidimensionales a través del motor ROLAP, que transforma dinámicamente sus consultas a consultas SQL. Éstas son ejecutadas sobre la base de datos relacional y con los resultados obtenidos se genera un conjunto multidimensional para devolver a los usuarios.

Este tipo de sistemas los datos detallados siempre estarán tan actualizados como los del Data Mart. La desventaja de ROLAP es que la recuperación de cualquier tipo de medida es más lenta que para otras arquitecturas.

En la arquitectura **MOLAP** (OLAP multidimensional) los agregados y una copia de los datos detallados se guardan en una base de datos multidimensional. Esta base de datos será la encargada del manejo, acceso y obtención de los datos solicitados. Con este tipo de base de datos se optimizan las consultas multidimensionales, pero requiere más espacio de disco.

Por último tenemos los sistemas **HOLAP** (OLAP híbrido), que combina ROLAP y MOLAP. Intenta tener las ventajas de ambos sistemas. Almacena el cubo y los agregados en una base de datos multidimensional, lo que permite una recuperación rápida de los agregados. Sin embargo, los datos detallados residen en una base de datos relacional. HOLAP sacrifica el tiempo de respuesta de los datos detallados para conseguir una menor latencia y mejorar los tiempos de carga.

1.2.4. Mondrian

Pentaho [2] es una compañía que ofrece soluciones tecnológicas para el análisis y manejo de datos. Estas soluciones open source están escritas en Java y hechas para ser utilizadas a través de aplicaciones escritas en el mismo lenguaje. Esto hace de Pentaho una buena solución para cubrir diversas necesidades[1] empresariales:

- Reporting: Generar informes de forma sencilla con diferentes contenidos (tablas, gráficos...).
- Análisis: Navegación a través de los datos (generalmente en forma de tablas), cambiado la información consultada y su nivel de agregación de forma rápida y dinámica.
- Data Mining: Extracción no trivial de información que reside de manera implícita en los datos.
- Integración de Datos: Extraer, reformatear, limpiar y cargar datos entre sistemas homogéneos.

Mondrian es una aplicación que pertenece al paquete Business Intelligence de Pentaho y está orientada al análisis de datos. Es un servidor OLAP, es decir, un intermediario entre una aplicación de consulta (escrita en Java en este caso) y una base de datos con los datos fuente. Se encarga de construir y mantener el cubo OLAP.

Un sistema que utiliza Mondrian se compone de cuatro capas: presentación, cálculo, agregación y capa de almacenamiento.

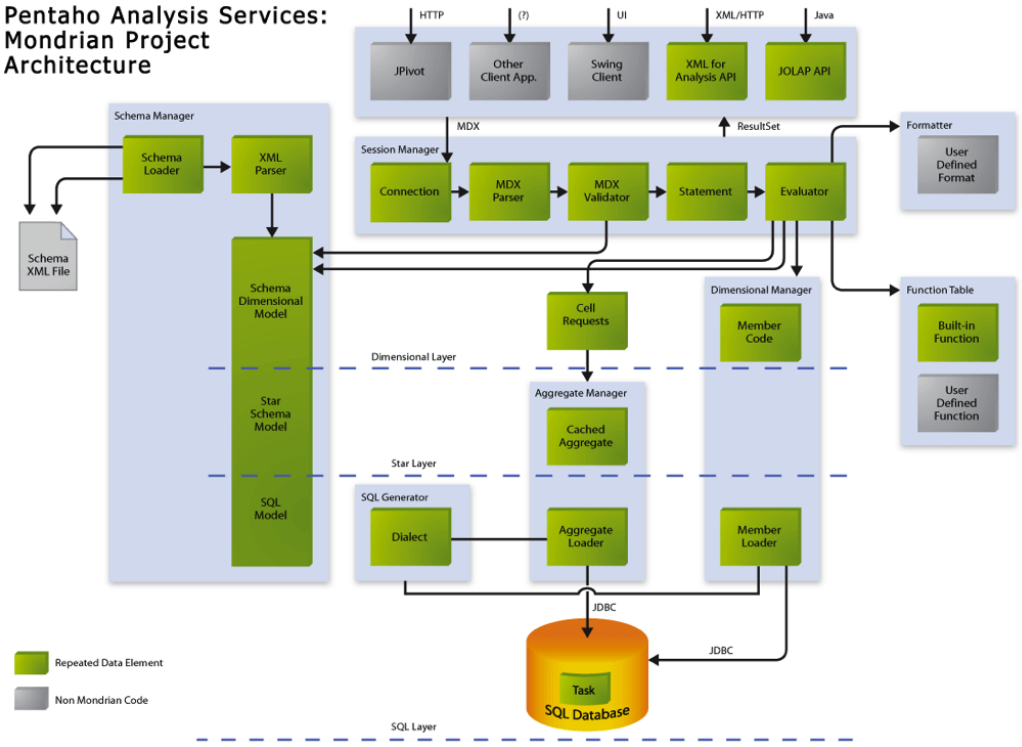


Figura 1.4: Estructura interna de Mondrian.

La capa de **presentación** (el objetivo de este proyecto) es la encargada de presentar los resultados al usuario final y le permite interactuar con la aplicación. Existen muchas formas de presentar estructuras de datos multidimensionales, incluyendo tablas pivotantes, mapas, gráficos de líneas, barras o porciones... Todas estas formas de representación tienen en común el uso de dimensiones, métricas o celdas de datos cuya información es obtenida a través del servidor OLAP. La interfaz por defecto que acompaña a Mondrian es Jpivot, un cliente JSP bastante sencillo.

La segunda capa (**cálculo**) es la encargada del análisis sintáctico, validación y ejecución de las consultas recibidas. El lenguaje de consulta que utiliza y "entiende" Mondrian es el MDX[3] ("Multi-Dimensional eXpressions") cuya sintaxis es similar a la del lenguaje SQL. Un ejemplo de consulta:

```
SELECT
   { [Medidas].[importe ventas]} on columns,
   { [Tiempo].[2008] }           on rows
FROM [cubo ventas]
WHERE ([Familia].[lacteos])
```

La consulta se evalúa en varias fases, primero se calculan los ejes y luego los valores de las celdas dentro de los ejes. Por eficiencia, las peticiones de celdas que se realizan a la capa de agregación se fragmentan en lotes. Desde esta capa se tiene acceso a la definición del cubo (metadatos) contenida en un fichero XML. Un ejemplo de archivo que define un cubo:

```
<Schema>
 <Cube name="Sales">
  <Table name="sales_fact_1997"/>
  <Dimension name="Gender" foreignKey="customer_id">
    <Hierarchy hasAll="true" allMemberName="All Genders"
        primaryKey="customer_id">
     <Table name="customer"/>
     <Level name="Gender" column="gender" uniqueMembers="
         true"/>
    </Hierarchy>
  </Dimension>
  <Measure name="Unit Sales" column="unit_sales"
      aggregator="sum" formatString="#,###"/>
 </Cube>
</Schema>
```

Estos metadatos son usados por Mondrian para mapear las consultas sobre el modelo dimensional hacia el modelo relacional. Esto quiere decir que Mondrian es un servidor "ROLAP" y por tanto realiza las peticiones de datos a un sistema gestor de base de datos relacional, a pesar de que su objetivo es hacer consultas sobre sistemas multidimensionales. Tener un sistema ROLAP nos facilita la creación de cubos si ya disponemos de una base de datos relacional donde tenemos almacenada la información a consultar. Mondrian se encargará de crear una capa virtual (con ayuda de otros mecanismos) para que podamos consultar sobre nuestro cubo sin tenerlo físicamente creado. En resumen, esta capa recibe instrucciones MDX y las transforma a SQL para realizar la adquisición de datos.

La tercera capa es la de **agregación**. Una agregación es un "resumen" de datos precalculados que mejora el tiempo de espera al tener preparadas las respuestas antes de que se planteen las consultas. Las agregaciones se pueden plasmar de forma física creando en la base de datos nuevas tablas y cargándolas con datos calculados a partir de los datos de origen. Esta capa se encarga también de mantener una memoria (cache) de datos utilizados recientemente por si son útiles en un futuro. Los dos mecanismos nombrados anteriormente (tablas agregadas y cache) son usados por Mondrian para agilizar el costoso proceso de consulta sobre cubos ROLAP.

Por último, tenemos el acceso a los datos (**almacenamiento**). Esta capa es la responsable de proporcionar los datos de las celdas obtenidos de un gestor de base de datos relacional o a través de un repositorio de datos desarrollado para funciones de almacenamiento multidimensional.

De estas capas, las dos intermedias (2 y 3), forman Mondrian que junto con el resto forman todo el sistema.

En general Mondrian ofrece las siguientes funcionalidades:

- Cuenta con un parseador y evaluador para consultas en lenguaje MDX. Se traduce de este al SQL de la base de datos que se esté usando.
- La definición del modelo dimensional se encuentra en un fichero XML, con una estructura determinada, al que Mondrian tiene acceso.
- Utiliza JDBC para la conexión con el sistema gestor de base de datos.
- Los resultados de las consultas dimensionales son almacenados en una memoria cache administrada por Mondrian.
- Mondrian trata de calcular los datos agregados con los campos que se encuentran en la cache para conseguir mejorar el rendimiento.

Capítulo 2

Objetivos

En este capítulo se describe el objetivo que se pretende alcanzar con este proyecto y se realiza un breve análisis de las aplicaciones ya existentes que realizan la misma función que la que se pretende desarrollar.

Índice

2.1. Descripción

El objetivo de este proyecto es desarrollar un cliente web para la exploración y visualización de cubos OLAP. Este cliente contará con diversos modos de exploración y visualización interactiva de los cubos multidimensionales (operaciones roll-up, drill-down, slice-dice, etc.). Entre estos modos de visualización existirá la posibilidad de representar la información de forma tabular y gráfica así como la opción de exportar los datos a un fichero plano. Como caso de estudio se utilizarán datos del sector sanitario, en concreto, datos de hospitalización de pacientes. Los cubos se construirán con la herramienta Mondrian de la suite Bussines Inteligence Pentaho (open source).

Aunque en el mercado comercial existen varios productos, estos son aún muy caros y por ello, el objetivo es crear un add-on reutilizable y de uso público para poder ser utilizado en cualquier RIA (aplicaciones de internet enriquecidas) mediante Java y el framework de Vaadin [13].

2.2. Análisis de alternativas

Algunos de los productos ya existentes que realizan las mismas funciones de visualización y navegación por cubos, se han probado antes de realizar este proyecto y se ha intentado captar las ventajas e inconvenientes de cada uno de ellos para tener una idea materializada de nuestro objetivo. A continuación se destacarán algunos aspectos de algunas de estas herramientas: jPivot, Pentaho Analayzer y Dundas.

2.2.1. jPivot

JPivot [10] es una librería de componentes JSP utilizada para construir tablas OLAP generadas de forma dinámica. El ejemplo que se ha probado viene asociado al servidor de cubos Mondrian, como cliente web de prueba para este servidor.

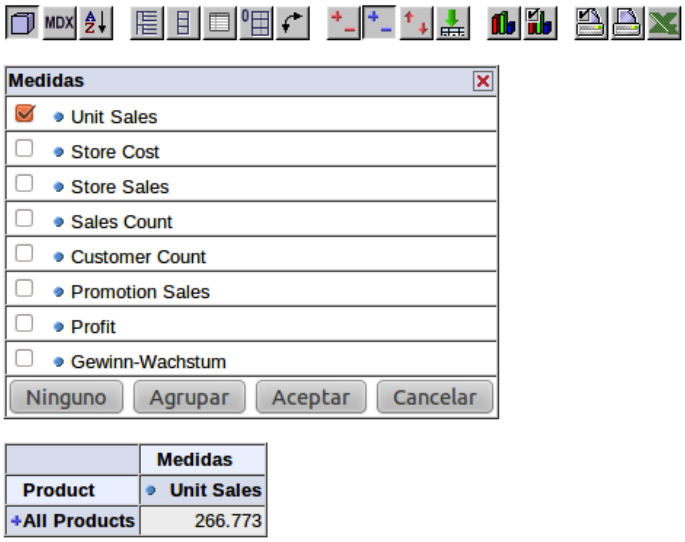


Figura 2.1: jPivot

Las principales ventajas de esta interfaz son su sencillez y el gran número de opciones que ofrece, así como su alto grado de configuración. Entre sus opciones más destacadas permite la ejecución de consultas MDX introducidas como texto por el usuario y una gran personalización en la visualización gráfica.

Las principales desventajas son su poco atractivo visual y una interfaz de elementos seleccionables poco práctica. Las tablas son muy sencillas, sin ningún tipo de efecto visual, igual que sus menús de opciones, todos los elementos muy sobrios. Por otra parte los menús para seleccionar los elementos a consultar, utilizan un mecanismo de opciones individuales (ver Figura 2.2) poco práctico para un uso continuo. Además de todo, esto no es posible

el cambio dinámico entre cubos de un mismo esquema.

Figura 2.2: Elementos seleccionables en jPivot

2.2.2. Pentaho Analayzer

Pentaho Analayzer [11] es el nuevo visualizador de cubos de la empresa Pentaho. Esta herramienta viene incluida dentro de los paquetes “Professional” y “Enterprise” que ofrece la empresa dentro de sus soluciones Business Analytics.

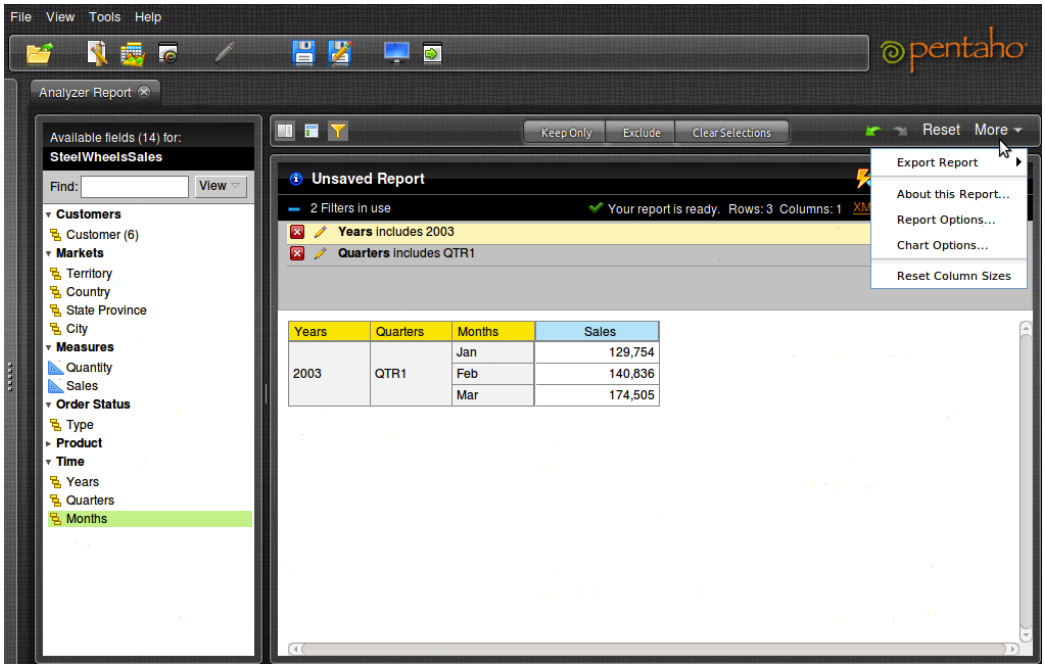


Figura 2.3: Pentaho Analyzer

Sus principales ventajas son su atractivo visual y su facilidad de manejo. Tiene un aspecto visual muy cuidado y cuenta con unos efectos de acabado muy buenos desde mi punto de vista. Se integra perfectamente con el resto de elementos de su entorno y al usarlo, da la sensación de estar ante una aplicación de escritorio en lugar de una aplicación web. Su usabilidad es muy buena y su manejo intuitivo, utiliza el Drag&Drop para realizar la selección, eliminación y ordenación de elemento consultados.

Sus principal desventaja es la imposibilidad de realizar una navegación sencilla dentro de los elementos consultados en forma tabular. Otra de sus desventajas reside en que no es gratuita ni de código abierto.

2.2.3. Dundas OLAP services

Dundas OLAP services[16] es el visualizador para cubos OLAP de la empresa Dundas y esta orientado a su utilización en tecnologías .Net.

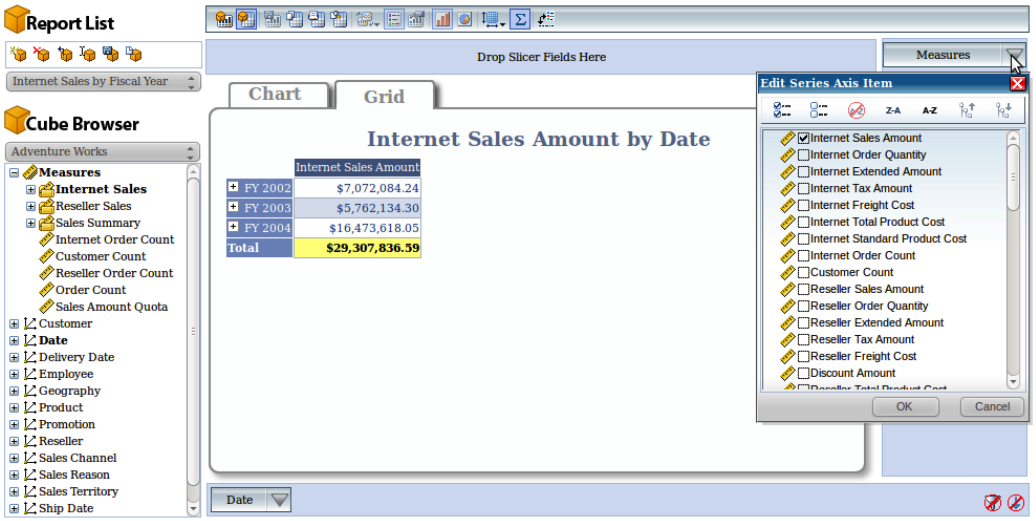


Figura 2.4: Visualizador Dundas

Como puntos fuertes de este visualizador de cubos, podemos destacar una buena distribución de sus elementos y la usabilidad a la hora de añadir elementos a consultar o filtrar los datos. También dispone de un amplio abanico de opciones de exportación y sorprende su capacidad para la navegación en modo gráfico, algo que ninguno de los visualizadores probados ofrece. La facilidad y sencillez de navegación recuerdan a las de jPivot pero con mayor atractivo visual y un mecanismo de selección de métricas y dimensión más práctico.

Como únicos "peros" a esta aplicación, o al menos a la demo [14] que he podido probar, es el desaprovechamiento del área de usuario, a pesar de tener gran cantidad de espacio libre en la ventana no lo aprovecha, y mantiene todos sus elementos de una forma muy compacta. Esta herramienta tampoco es de código libre ni es personalizable.

Capítulo 3

Planificación del Proyecto

En este capítulo se describe la metodología a seguir durante la fase de diseño y desarrollo de la aplicación. Al ser una metodología ágil se realiza una breve introducción de las mismas.

Índice

3.1. Metodologías Ágiles

En los comienzos del desarrollo de software el tiempo de vida de un producto acabado era muy largo y durante ese tiempo generaban beneficios a las empresas que lo desarrollaron. Sin embargo a partir de los 80 la vida de los productos pasó a ser cada vez más corta y las empresas tenían que adaptarse a los cambios continuos y ser capaces de ofrecer novedades continuamente.

Esto era un problema para las empresas que hasta entonces utilizaban metodologías de desarrollo clásicas, obtenidas de otras ramas de ingeniería, como la metodología en cascada. En esta se debían seguir una serie de etapas, o fases, consecutivas, en las que se definían un conjunto de actividades con unas metas. Típicamente las fases son: análisis, diseño, codificación, prueba y mantenimiento. Esta metodología tiene una ventaja, su sencillez, sin embargo no tiene en cuenta el contexto y es bastante inflexible a cambios.

No es hasta 2001 que a través del manifiesto ágil[17] se propone una alternativa al uso de las metodologías clásicas. Una serie de expertos y profesionales del mundo del software proponen una serie de "principios" que había que tener en cuenta para mejorar la manera de desarrollar software:

- Individuos e interacciones

- Software que funciona
- Colaboración con el cliente
- Responder ante el cambio

Se pretendía con esto desarrollar nuevas metodologías (ágiles) que tuvieran la capacidad de adaptarse al cambio, que tuvieran más en cuenta los requisitos del cliente, entregas de software funcional cada poco tiempo, equipo de personas motivado y trabajando juntos con buena comunicación, simplicidad y cuya máxima prioridad fuera satisfacer al cliente con entregas tempranas y continuas.

Por tanto, el agilismo es una respuesta a una necesidad, la habilidad para adaptarse al cambio. No pretende ser el perfeccionismo, simplemente se tiene en cuenta que el software es propenso a errores por naturaleza y la idea es tomar medidas para minimizarlos desde el principio. También existen etapas como en la metodología clásica, pero en este caso están solapadas, en lugar de consecutivas, y se llevan a cabo repetidas veces (de forma iterativa) en las que se pueden añadir requisitos, modificar diseño, añadir funcionalidad... adaptación.

3.2. XP y TDD

Para este proyecto se ha decidido seguir la metodología ágil XP (eXtreme Programming) [design] con la técnica de diseño e implementación de software TDD (Test Driven Development).

Algunos de los valores que XP pretende poner en juego son: la comunicación, la simplicidad, retroalimentación y coraje para llevar a cabo decisiones relacionadas con los anteriores valores.

Las historias de usuario son la técnica utilizada para especificar los requisitos del software. El cliente escribe brevemente las características que el sistema debe poseer, estarán escritas en el lenguaje de su negocio. Se deben evitar las ambigüedades y por ello en lugar de redactar las características se pueden definir con ejemplos. La idea es tener una serie de elementos que puedan ser descompuestos en tareas de programación y asignados a los programadores para ser implementados durante una iteración.

Los roles más destacados utilizados en esta metodología para este proyecto son:

- Programador: Escribe las pruebas unitarias y produce el código del sistema.
- Cliente: Escribe las historias de usuario y les da prioridades para decidir cuáles se implementan en cada iteración.

Como se ha dicho, junto con XP se va a utilizar TDD, desarrollo dirigido por test. El algoritmo TDD sólo tiene tres pasos:

- Escribir la especificación del requisito (el ejemplo, el test).
- Implementar el código según dicho ejemplo.
- Refactorizar para eliminar duplicidad y hacer mejoras.

En primer lugar se parte de un requisito (o historia de usuario) escrito por el cliente y lo expresamos en código, es decir, escribimos un test para nuestro software. Este test, en la medida de lo posible, se encargará de probar una sola historia de usuario, y su nombre debe ser autoexplicativo. Hay que tener en cuenta que el código que va a validar el test aún no está desarrollado. Es en el siguiente paso de la etapa cuando se desarrolla el código mínimo para que ese test pase. El código desarrollado debería cumplir con unos principios de diseño establecidos, como por ejemplo S.O.L.I.D.[18] (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion). Por último hay que refactorizar el código escrito tanto el de la aplicación como el de los test y así conseguir mejorar el diseño sin cambiar el comportamiento.

Al diseñar y codificar las pruebas ya estamos (sin darnos cuenta) trabajando en el diseño del código, pues establecemos los nombres de las clases/métodos, estructura de constructores, parámetros de los métodos... Y todo esto es lo que vamos a necesitar porque está basado en las historias de usuario, sin funcionalidades extra. Además orienta a que todo el código desarrollado tenga la posibilidad de probarse.

Capítulo 4

Tecnología usada

En este capítulo se describen las herramientas y tecnologías utilizadas para la creación de la aplicación.

Índice

4.1. Herramientas

Como principal herramienta para llevar a cabo este proyecto se ha utilizado el IDE (integrated development environment o entorno de desarrollo integrado) de programación **Eclipse**[19].

Puesto que el diseño y desarrollo de la aplicación está basado en pruebas hemos necesitado una herramienta que nos de soporte para codificarlas y posteriormente ejecutarlas sobre el código desarrollado. La herramienta elegida ha sido **jUnit**[junit1] que se integra perfectamente con nuestro entorno de programación.

JUnit[20] es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck (quién popularizó TDD) que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. Es por tanto un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Básicamente, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que retornó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

Como se ha visto en la sección 1.2.4 **Mondrian** nos servirá de enlace entre nuestra aplicación y la base de datos donde resida nuestro cubo OLAP.

Se incluirá en nuestro proyecto como una librería y podremos enviarle peticiones en forma de consulta de texto, y este nos devolverá los datos solicitados.

También se ha utilizado otro módulo de Pentaho, **Cube Designer**. Esta herramienta nos permite definir los cubos OLAP mediante una interfaz gráfica y posteriormente se encarga de generar el archivo XML con la estructura de dicho cubo.

Como gestor de dependencias se ha usado **Maven**[12]. Maven es un framework de gestión de proyecto de software, que proporciona un modelo estándar de gestión y descripción de proyectos. Será el elemento encargado de obtener las librerías de las que depende nuestro proyecto a través de los repositorios que definamos.

4.2. Vaadin

4.2.1. Origen y estructura interna

Vaadin[13] es un framework para el desarrollo de aplicaciones web, que se conocen como RIA (Rich Internet Applications), escritas en Java. Esta herramienta esta desarrollada por Vaadin Ltd, una empresa finlandesa de software especializada en el diseño y desarrollo de RIAs. Esta empresa ofrece a sus clientes servicios de planificación, implementación y soporte para proyectos software. Este framework fue desarrollado inicialmente para uso interno en la empresa, pero dado su potencial se ha convertido en su producto estrella, y además de hacerlo de código libre, ofrecen soporte y desarrollo comercial para él.

Consta de dos partes, un framework para el lado del servidor y un motor del lado del cliente que se ejecuta en el navegador como un programa JavaScript, renderizando la interfaz del usuario y encargandose de las interacciones entre el usuario y el servidor.

Esta herramienta esta orientada a la programación del lado del servidor, es decir, permite que te olvides de que el resultado será un elemento web y puedas desarrollar tu interfaz como una aplicación Java de escritorio, es decir, como si estuvieras usando AWT, Swing o SWT, de una forma muy sencilla.

Vaadin será el encargado de crear los componentes visuales en el navegador, así como la comunicación AJAX entre él y nuestro servidor. Esto hace que no sea necesario aprender tecnologías web como HTML o JavaScript. Todo esto hace que la parte web sea invisible y el programador se pueda centrar en la lógica de la aplicación, pasando el navegador web a ser una plataforma sobre la que mostrar nuestra interfaz.

4.2.2. GWT

El núcleo de Vaadin utiliza GWT[21] (Google Web Toolkit), para renderizar la interfaz de usuario. GWT es un entorno de desarrollo Java, basado en software libre, y que permite escribir aplicaciones AJAX fácilmente. Permite codificar las aplicaciones en el lenguaje de programación Java y luego se encarga de compilarlo, traduciendo la parte del cliente a lenguaje de programación JavaScript + HTML + CSS, generando código JavaScript más eficiente que escrito a mano y evitando que nosotros tengamos que desarrollar esa ardua tarea.

Tener la posibilidad de programar en Java conlleva muchas ventajas, como poder utilizar los IDE existentes para este lenguaje, como en este caso Eclipse, así como sus herramientas de depuración.

Las aplicaciones generadas por GWT ejecutan código Java del lado del servidor y se encargan de las comunicaciones entre el cliente y el servidor utilizando llamadas asíncronas.

4.2.3. Arquitectura

Una aplicación Vaadin se ejecuta como servicio en un servidor web Java (como Tomcat o WebSphere) atendiendo peticiones HTTP. Existe un módulo que se encarga de recibir las peticiones de los clientes a través de ese servidor, las interpreta y convierte en eventos de usuario para una sesión web determinada. A su vez ese evento se encuentra asociado a un componente de la interfaz de usuario.

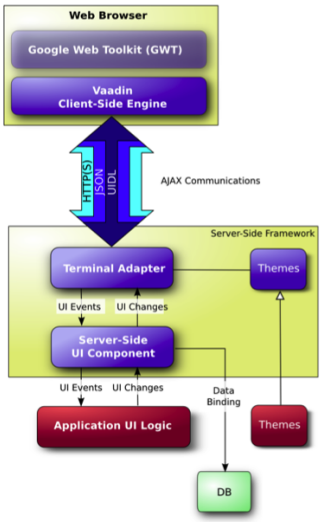


Figura 4.1: Arquitectura Vaadin (Fuente "El libro sobre Vaadin").

A continuación se mostrará una breve descripción de los diversos elementos que compo-

nen Vaadin:

- Componentes: La interfaz de usuario se forma mediante componentes que son creados y distribuidos en nuestra aplicación. Cada uno de ellos está compuesto por una parte código de cliente y una de servidor. La parte del servidor se encarga de serializar los elementos y enviarlos al navegador. La parte del cliente es capaz de serializar las interacciones de los usuarios y mandarlas a la aplicación para ser manejadas como eventos en la parte del servidor. A continuación esos eventos son trasmitidos a la lógica de la aplicación.

- Motor en la parte del cliente: Se encarga de la renderización de componentes en el navegador web usando GWT y de la comunicación mediante peticiones y respuestas asíncronas mediante HTTP o HTTPS.

- Temas: Son usados para dar estilo a la parte visual de la aplicación, están definidos como CSS. Vaadin proporciona tres temas por defecto, aunque también se pueden crear nuevos.

  Desde el punto de vista del programador, se dispone de una aplicación que desciende de "com.vaadin.Application" y mediante componentes, layouts para organizarlos y manejadores de eventos de los dos anteriores, se puede diseñar y desarrollar la parte visual y comportamiento (la lógica) de nuestra aplicación web.

4.2.4. Vaadin como opción

Por todo esto, este framework, está pensado para desarrollar aplicaciones web de forma rápida y sencilla, basándose en la idea de construcción por componentes y dando la posibilidad de desarrollar tu aplicación como una composición de módulos ya programados.

Desde la empresa se prevé un gran futuro para esta herramienta y se apuesta por ella. A pesar de que la comunidad que hay detrás aún no es muy numerosa, cuenta con el apoyo y desarrollo por parte de una compañía, lo que le da un voto a favor.

En esta arquitectura basada en componentes es donde entra este proyecto y su idea de construir un nuevo componente fácilmente reutilizable para la visualización y navegación de cubos OLAP, definiendo por un lado la lógica de navegación en la parte del servidor y por otro una atractiva interfaz que se comunique con la lógica y pueda ser personalizable cambiando el tema de la aplicación que la contiene.

Capítulo 5

Presentación del problema y análisis de requisitos

En este capítulo se presenta el caso de estudio concreto sobre el que se va a desarrollar y probar la aplicación, además se definen los requisitos o historias de usuario que tendrá que cumplir la aplicación.

Índice

5.1. Caso de estudio CMBD

La mayoría de requisitos mostrados en el siguiente apartado, están más orientados a ser historias de usuario, que como ya hemos dicho, deben contener ejemplos para evitar ambigüedades. Los ejemplos en este caso están basados en un cubo con datos de hospitalizaciones médicas obtenidos del CMBD [6] (Conjunto Mínimo Básico de Datos) de 2001. Estos datos se encuentran disponibles en [5].

La base de datos de CMBD es un registro de altas de hospitalización, que además de servir de utilidad para numerosos investigadores, sirve para la generación de estadísticas de referencia estatal que son usadas para diferentes fines [7].

5.1.1. Cubo OLAP de Hospitalizaciones

Sobre los datos del CMBD se ha creado un cubo OLAP para permitir un acceso a esa información de manera más sencilla y rápida.

Con este cubo podemos obtener los datos sobre el número de casos, la estancia media de los pacientes y el coste medio incurrido. Esas son las métricas que se han considerado relevantes y se han incluido en este cubo.

Por otra parte, podemos consultar esos datos teniendo en cuenta la localización del hospital, el grupo al que pertenece ese hospital, el sexo de los pacientes, el tipo de ingreso (programado o no programado), la edad de los pacientes (por tramos quinquenales) o sus diagnósticos. Estas son algunas de las dimensiones que se pueden extraer de la base de datos para incluirlas en nuestro cubo.

5.2. Requisitos

En este caso, al realizar el proyecto en una empresa, el responsable de la misma es quien tendrá el rol de cliente. Él será el encargado de definir los requisitos y validar los resultados obtenidos, pudiendo así modificar o añadir requisitos durante el desarrollo del proyecto.

La empresa, entre otros campos, se dedica a desarrollar soluciones personalizadas dentro del **Bussines Intelligence**. Para el desarrollo de estas soluciones están usando cada vez más el anteriormente mencionado framework de **Vaadin**, y dadas estas dos situaciones surge la necesidad de este proyecto: desarrollar un componente reutilizable para la navegación de cubos OLAP.

Los requisitos para este nuevo componente atendiendo a otras aplicaciones ya realizadas y a las necesidades de la empresa son los siguientes:

- Posibilidad de elegir cubo de todos los disponibles en el repositorio multidimensional que se está utilizando (esquema xml en este caso particular de Mondrian).
- Al seleccionar un cubo se deben mostrar todos los elementos del mismo
- Añadir y quitar Métricas sobre las que consultar
  - Al añadir la métrica “Casos” en las filas no se muestra ningún valor
  - Al añadir la métrica “Casos” en las columnas se muestra el número de casos total
  - Al quitar la métrica “Casos” de las columnas desaparecerá el número de casos total
  - Al añadir una métrica puedo elegir en que posición va a ser añadida
- Añadir y quitar Dimensiones sobre las que consultar
  - La dimensión “CIE9MC” (padre) que no es consultable no se permitirá añadir
  - Si añadimos una sola dimensión en las filas o las columnas, no se muestra ningún valor
    - Si añado solo “Tipo Ingreso” en filas no se muestra nada.

    - Si añado solo “Tipo Ingreso” en columnas no se muestra nada.
  - Por muchas dimensiones que se añadan, si no se añade ninguna métrica, no se muestra ningún valor.
  - Si añado “Sexo” en filas, no puedo añadirla en las columnas
  - Si añado “Sexo” en columnas, no puedo añadirla en filas
  - Si añado “Sexo” en filas o columnas, no puedo añadirla en filtro
  - Al añadir una dimensión puedo elegir en que posición va a ser añadida
    - Puedo añadir “Tipo Ingreso” en primer lugar estando ya añadida “Sexo”
- Añadir y quitar filtro seleccionando dimensión y valor de filtrado (operación *slice*)
  - Al añadir “Geografía Hospital” como filtro, se puede seleccionar “Cantabria” como filtro
  - Al seleccionar “Cantabria” como filtro, los datos que se están mostrando se filtran automáticamente
  - Al seleccionar “Cantabria” como filtro, las siguientes consultas mostrarán solo los datos de Cantabria
  - Al eliminar el filtro “Cantabria” los datos mostrados pasarán a ser los de toda la “Geografia Hospital”
- Cambiar de cubo
  - Al seleccionar un cubo diferente si hay datos de consultas anteriores dejarán de mostrarse
- Exportar datos consultados
  - Si se están mostrando todos los “Casos” y se exportan, se creará un documento excel en el que aparecerá en el nombre de la columna “Casos” y como valor “3297074”
- Alternar entre vista gráfica y tabular
  - Cuando se solicite representar gráficamente los datos, se mostrará una gráfica con los últimos datos válidos mostrados en la tabla
  - Si al estar visualizando la gráfica se solicita “Pivotar” entre filas y columnas, la gráfica reflejará el cambio
- Cambiar nivel de agregación de las dimensiones consultadas (*drill-down* y *roll-up* accionado sobre las dimensiones)
  - Al interactuar con el elemento “TodosLosSexos” frente a “Estancia Media” se mostrarán los datos de “Estancia Media” por cada tipo de sexo.

  - Al interactuar con el elemento "TodosLosSexos" con todos los tipos de sexo mostrados frente a "Estancia Media", se mostrarán los datos sumarizados y desaparecerá el detalle, quedando "TodosLosSexos" frente a "Estancia Media"

- Al seleccionar una dimensión en filas y una métrica en columnas se realizará la consulta más sumarizada que contiene a los dos elementos
  - Al seleccionar "Tipo Ingreso" en filas y "Estancia Media" en columnas, se debe mostrar que la estancia media de todos los ingresos es de "7,77"
  - Al seleccionar "Tipo Ingreso" y "Sexo" en filas y "Estancia Media" en columnas, se debe mostrar que la estancia media de todos los ingresos y todos los sexos es de "7,77"
- Posibilidad de intercambiar las filas y las columnas consultadas (*pivotar*)
- Ordenación de los valores consultados en columnas
- Posibilidad de mostrar los datos consultados de forma gráfica
  - Mostrar los datos consultados en forma de gráfico de barras
  - Mostrar los datos consultados en forma de gráficos de porciones
  - Permitir elegir el tamaño de los gráficos
  - Se mostrará un gráfico circular por columna
  - Se mostrará una barra por fila y un grupo de barras por cada columna
- Permitir establecer todos los textos y mensajes estáticos que son mostrados por la aplicación, es decir, permitir la internacionalización (i18n[1]).
- Traducción de los valores devueltos por una consulta al cubo.
  - Puedo elegir que ha de mostrarse en lugar de "1", por ejemplo "Masculino", al hacer drill-down sobre "TodosLosSexos" .
  - En lugar de mostrar "Geografía Hospital" en las cabeceras puedo establecer que se muestre "Comunidad".

[1] Numerónimo para la internacionalización.

Capítulo 6

Diseño e implementación de la solución propuesta

Este capítulo hace un recorrido por el diseño y construcción de los diferentes módulos de la aplicación partiendo de los requisitos. También se detalla el diseño de la interfaz y sus diferentes componentes. Por último muestra el resultado final del componente desarrollado.

Índice

6.1. Conexión con Mondrian y estado de consulta

Partiendo de los requisitos planteados anteriormente, se ha tomado como primera fase de diseño la relativa al listado y selección del cubo a consultar.

La primera prueba unitaria codificada trata de obtener una lista de nombres de cubos extraídos del esquema XML pasado como parámetro. Esto nos lleva a codificar una clase que llamaremos "Estado" con un constructor que recibirá la ruta del esquema. También habrá un método que devolverá una lista con los nombres de los cubos disponibles en el esquema. La información contenida en el esquema XML ha de ser leída por Mondrian, por tanto crearemos otra clase, que llamaremos "ConexionOlap", y será la encargada de la comunicación de nuestra aplicación con el servidor de cubos. Ahora el constructor de la clase Estado, creará una instancia de "ConexionOlap" para poder extraer de ella toda la información del esquema. El método que habíamos creado anteriormente, devolverá la lista de nombres de cubos obtenida de su instancia de "ConexionOlap".

A continuación creamos otra prueba que intente seleccionar un cubo con un determinado nombre pasado por parámetro. Podemos comprobar el éxito de la prueba porque el método devolverá verdadero o falso dependiendo de si la selección se ha llevado a cabo correctamente. Para que este test pase se crea el método correspondiente en la clase "Estado" que comprueba que el nombre del cubo solicitado está dentro de los cubos disponibles en el esquema, y de ser así, almacenará en una variable este nombre, quedando seleccionado este cubo.

Hasta aquí queda codificada la primera parte que nos permite listar y seleccionar un cubo, cubriendo así el primer requisito.

El siguiente test unitario a construir está relacionado con los anteriores, pues tiene que ver con lo que sucede después de seleccionar un cubo. Tenemos que ser capaces de obtener la información del cubo seleccionado, es decir, obtener la lista de sus métricas y dimensiones. Separaremos este requisito en dos pruebas diferentes, por un lado probaremos la obtención de una lista con las métricas disponibles y en otra prueba intentaremos obtener la lista de dimensiones. Esto nos lleva a crear dos nuevos métodos en la clase de "Estado", que se encargarán de obtener la información solicitada a la pasarela con Mondrian, es decir, a través de la instancia de "ConexionOlap".

Una vez que tenemos disponibles la lista de elementos consultables necesitamos tener la posibilidad de seleccionar algunos de ellos. Al crear los test y el código que lleva a cabo la selección de una métrica o dimensión se observa que el comportamiento es el mismo y tenemos bastante código repetido, es hora de refactorizar, en este caso unificando.

Aunque en los requisitos se hace distinción entre añadir métricas o dimensiones, se ha creado una única prueba que servirá para añadir elementos en general. Al igual que antes, comprobaremos la efectividad del método teniendo en cuenta el parámetro (verdadero o falso) que devuelve. Una vez creado el test que invoca un método que tiene como parámetro

el nombre del elemento y si se quiere seleccionar en filas o columnas, se realiza la codificación del método en la clase "Estado". Este método comprobará que el elemento indicado forma parte de los elementos consultables, y de ser así, lo almacenará en una lista de elementos seleccionados, teniendo en cuenta ahora si es una métrica o una dimensión y diferenciando entre filas y columnas. Resumiendo, se comprueba si es un elemento del cubo sobre el que podemos hacer consultas y se almacena en la lista correspondiente (métricas/dimensiones en filas/columnas).

Tenemos que tener en cuenta ahora el conjunto de ejemplos (historias de usuario) que explican el comportamiento de la adición tanto de métricas como dimensiones. Para ello crearemos un test por cada ejemplo y esto nos llevará a tener que modificar el comportamiento interno, con una serie de bloques condicionales, del método desarrollado en el apartado anterior.

El comportamiento de la selección de filtros es similar al de los elementos consultables, sin embargo, necesitamos obtener los hijos de la dimensión seleccionada como filtro para poder seleccionar uno (o varios) como valor a filtrar. Creamos el test correspondiente que intenta añadir un filtro pasando como parámetro el nombre de una dimensión, se espera que devuelva una lista con los valores (hijos) disponibles para esa dimensión. Al igual que antes, al codificar necesitamos comprobar que es una dimensión válida, que no esté ya seleccionada en filas o columnas, e intentaremos obtener sus hijos a través de la pasarela "ConexionOlap".

Al seleccionar un cubo distinto al actual debe desaparecer la información del anterior y los elementos que habíamos consultado sobre él. Para ello creamos un test en el que seleccionamos un cubo, realizamos unas selección de elementos sobre el y a continuación seleccionamos un cubo distinto dentro del esquema. Para poder comprobar si se han eliminado correctamente los elementos seleccionados del cubo anterior, necesitamos crear métodos que nos permitan obtener la lista de elementos (métricas/dimensiones) seleccionados en filas y en columnas, así como los filtros. El método para cambiar de cubo ya lo tenemos codificado, es el mismo que para la selección inicial del cubo, aunque ahora además será el encargado de borrar todos los elementos consultables y filtros seleccionados anteriormente.

6.2. Consultas MDX

De momento tenemos un módulo que nos permite definir un estado deseado con el objetivo de recibir la información relacionada con él. Para poder realizar las peticiones oportunas a nuestro servidor de cubos, necesitamos un módulo que transforme ese estado en una consulta MDX. A este módulo le llamamos "MdxQuery".

El objetivo de seleccionar métricas y dimensiones es obtener a cambio una serie de datos. Para ello creamos un test en el que tras seleccionar una dimensión en las filas y una métrica en columnas, se invoca otro método para obtener posibles resultados de la consulta realizada con esa selección. Esto nos lleva al diseño del formato de respuesta de esos datos. Con el

objetivo de que sea lo más versátil y reutilizable posible se ha diseccionado el resultado en diferentes componentes: cabeceras de las columnas, cabeceras de las filas y valores. Pasamos a codificar los métodos que devuelven cada uno de estos elementos.

CABECERA FILA	CABECERA COLUMNA
+ Valor	950.000

Figura 6.1: Partes diferenciadas de los resultados.

Escribimos un test en el que crearemos un estado con la métrica "Casos" seleccionada en las columnas, esto debería devolvernos un valor con el número de casos total en nuestro cubo de Hospitalizaciones. Un método "Consulta" será el encargado de pedir al módulo "MdxQuery" la consulta necesaria para obtener los datos, pasándole las variables de estado (elementos seleccionados en filas y columnas) como parámetros, y esta consulta tendrá que ser ejecutada sobre el servidor de cubos a través del módulo "ConexionOlap". De este último módulo obtenemos los datos en un formato multidimensional definido por Mondrian, tenemos por tanto la necesidad de crear otro método que se encargue de extraer los datos de esa estructura e introducirlos en la estructura que hemos definido anteriormente. De esta manera ya obtenemos el número de "Casos" a través del estado definido anteriormente.

Para poder recoger toda la información de cada elemento se necesita una clase que nos permita establecer y consultar diferentes parámetros que definirán el comportamiento de cada dato devuelto. A esta clase la llamamos "CeldaCubo" y hablaremos de ella en el siguiente apartado 6.3 .

Mediante otro test definimos el comportamiento del creador de consultas en el caso de que la métrica "Casos" se seleccione en filas y nada en las columnas. Dada esta situación el método que crea las consultas devolverá una vacía. Del mismo modo creamos test para añadir la dimensión "Tipo Ingreso" en filas y en columnas, y el resultado debería ser el mismo, una consulta nula.

El estado también tiene que permitirnos intercambiar las filas y columnas seleccionadas. Para ello creamos un test en el que definimos un estado con "Sexo" en las filas y "Casos" en las columnas. Al invocar el método "pivotar" del estado se espera obtener el mismo valor para el número de casos pero mostrando el "Sexo" en las columnas y la métrica "Casos" en las filas. Para pasar ese test necesitamos codificar el método que se encargará de intercambiar los elementos del estado seleccionados en filas por los seleccionados en columnas.

Al añadir un filtro al estado, la consulta realizada debería modificarse creando un "slice" del cubo. Para conseguir esto creamos un test en el que, al igual que en el apartado anterior, definimos un estado en el que consultamos el número "Casos" y añadimos ahora un filtro con la dimensión "Geografía Hospital" y con el valor "Cantabria", esto debería devolvernos un valor menor que el caso anterior. Para ello codificamos un método que al recibir una orden de filtrado se lo comunique al módulo "MdxQuery" que deberá añadir un filtro en la consulta MDX que se le solicite la próxima vez. Si le solicitamos al estado que consulte, obtenemos ahora un resultado menor que el anterior, los datos se han filtrado.

Necesitamos tener ahora el comportamiento contrario, al eliminar un filtro se vuelve a los resultados no filtrados. Creamos un test con el mismo estado anterior y al eliminar el valor filtrado el resultado debería ser mayor que el anterior e igual al inicial. Para ello nos sirve el mismo método anterior pero con unas pequeñas modificaciones, ahora le pasaremos como parámetro el estado de los filtros (actualizados) y el módulo "MdxQuery" los guardará en una variable, esto nos lleva a crear otro método que recibe como parámetro esa variable y lo convierte a un texto que contiene concatenados todos los valores de filtrado. Por tanto ahora para añadir o eliminar un filtro basta con invocar a este método con el nuevo estado de los filtros (indicando dimensión/es y valor/es por los que filtrar).

Pasamos a programar los test necesarios para la navegación por el cubo, es decir para el cambio en la agregación de los datos. El primero de los test parte de un estado en el que tenemos seleccionada la dimensión "Sexo" en las filas y la métrica "Estancia Media" en las columnas, realizando la consulta obtenida con ese estado obtenemos dos elementos "TodosLosSexos" y un valor de estancia media. La celda del primer elemento nos indica que es un elemento agregado sobre el que podemos interactuar y desagregar. Si queremos obtener la consulta necesaria para visualizar esa desagregación, no nos sirve el método anterior para crear consultas, pues ahora esa consulta no depende solo de los elementos (métricas y dimensiones) seleccionados, si no también de los datos consultados anteriormente, es decir, del resultado de la consulta anterior. Esto nos lleva a crear en "MdxQuery" un nuevo método que nos sirva para navegar por las agregaciones y nos devuelva la consulta apropiada, recibiendo como parámetro los resultados devueltos en la consulta anterior, en forma de "Celdas", indicando en ellos las operaciones a realizar (dril-down o roll-up).

6.3. Celda

A lo largo del desarrollo de la aplicación, se ha necesitado una serie de información acerca de los valores devueltos por las consultas con el fin de mostrarlos o poder crear las consultas correctamente. Esto ha llevado a la construcción de un objeto que recogiera todas las propiedades relevantes proporcionadas por Mondrian, como son las siguientes:

- Texto mostrado (puede ser un valor numérico)

- Un identificador único
- Indicador para saber si es un elemento agregado sobre el que podemos navegar
- Indicador para saber si es un elemento desagregado
- Nivel de jerarquía del elemento

Esas características son establecidas a través del constructor y métodos de la clase "CeldaCubo". Al refactorizar extraemos el comportamiento de esa clase en la interfaz "ICelda" que será implementada por "CeldaCubo".

6.4. Exportación de datos

Otro de los requisitos consiste en tener la posibilidad de exportar los datos consultados. Para conseguir esto desarrollamos un test que crea un estado en el que se selecciona la métrica "Casos" en las columnas, a continuación se invoca sobre ese estado el método exportar. Ese método debería devolvernos el fichero creado en forma de recurso. Para pasar ese test necesitamos codificar el método "exportar" dentro de "Estado". Este método utilizará otro módulo que llamaremos "GeneradorExcel", y que a través de la librería "Poi" de Apache [8] tendrá la capacidad de generar archivos XLS. Este módulo recibirá las filas, columnas y valores, y devolverá un objeto File que contendrá esos datos para visualizarlos en forma de hoja de cálculo.

6.5. Interfaz gráfica de usuario

Una vez desarrollado el núcleo de la aplicación pasamos a diseñar y desarrollar la interfaz que servirá como puente entre ese núcleo y el usuario.

Algunos de los elementos que van a formar parte de la interfaz están ya prácticamente definidos: un selector con el nombre de los cubos, una lista de los elementos que contiene el cubo seleccionado, tabla o gráfica para mostrar los resultados de las consultas y zona donde seleccionar las métricas o dimensiones a consultar o filtrar. Sin embargo estos elementos son muy generales y no se detalla el comportamiento exacto, ni el tipo de componente por el que serán representados. Por tanto, entramos en detalle a continuación.

6.5.1. Selector cubo

Un elemento desplegable nos permitirá elegir uno de los cubos disponibles dentro del esquema con el que estemos trabajando. Este elemento será cargado con el nombre de los cubos al iniciar la aplicación, una vez leído y parseado el esquema XML. Al seleccionar

uno de los cubos se desencadenará un evento que utilizaremos para obtener los elementos disponibles (métricas y dimensiones) del mismo.

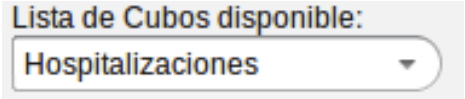


Figura 6.2: Selector de cubos disponibles.

6.5.2. Listado de elementos del cubo seleccionado

Los elementos disponibles del cubo tienen que ser listados pero teniendo en cuenta la división entre métricas y dimensiones. Como algunas de las dimensiones pueden tener una estructura jerárquica (subelementos consultables), se ha optado por asignar una estructura de árbol. Este árbol contendrá dos elementos de nivel superior que marcarán la separación entre métricas y dimensiones. Dentro de las dimensiones habrá algunos elementos que podrán ser expandidos mostrando así otro nivel de jerarquía dentro de esa dimensión.

Este árbol será rellenado con los elementos del cubo una vez seleccionado este. Los distintos elementos son obtenidos a través del módulo "Estado", con los métodos ya codificados que nos permiten obtener por un lado las dimensiones disponibles y por otro las métricas disponibles.

Figura 6.3: Elementos disponibles en el cubo seleccionado.

Al igual que pasó con las celdas de datos, ha sido necesario crear una nueva clase para albergar ciertas propiedades de los elementos que forman parte del árbol. Esta clase se denomina "NodoLista" y nos permite diferenciar entre métricas y dimensiones además de poder separar entre el texto mostrado y un identificador único.

6.5.3. Filas y columnas seleccionadas

Necesitamos de algún mecanismo que nos permita elegir, del árbol anterior, dos conjuntos distintos de elementos. Una vez elegidos se deberán listar y se dará la posibilidad de revocar esa selección.

En este caso se ha optado por asignar unas tablas como elemento gráfico para mostrar las listas de elementos seleccionados. El título de la tabla diferenciará entre elementos para las filas y para las columnas, y cada fila (o entrada) de la tabla, indicará una métrica o dimensión seleccionada del conjunto de las disponibles.

El mecanismo elegido para la selección ha sido el Drag&Drop, es decir, arrastrar elementos disponibles del árbol y soltarlos sobre la tabla correspondiente. Esta forma de selección no es muy intuitiva inicialmente para el usuario, pero una vez utilizada es muy útil y le da un gran atractivo visual a la aplicación.

Utilizaremos también el mecanismo anterior para quitar uno de los elementos seleccionados de las filas o columnas. En este caso, deberemos soltar el elemento sobre un componente especial que actuará de "sumidero". Al soltar sobre él una métrica o dimensión dejará de estar seleccionada (consultada) y podremos elegirla de nuevo. Este componente estará representado por la imagen de una papelera.

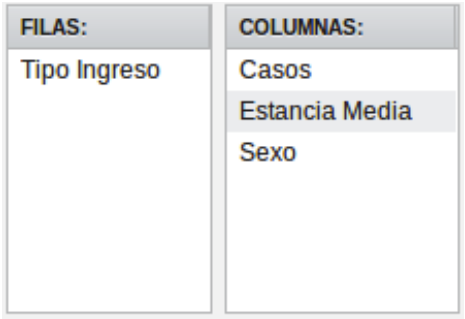


Figura 6.4: Elementos seleccionados en filas y columnas

6.5.4. Filtros seleccionados

Para el caso de los filtros utilizaremos también una tabla para listar los elementos filtrados y Drag&Drop para seleccionar los mismos. Sin embargo en este caso cada elemento seleccionado (cada fila de la tabla) estará representado por un botón. Este botón nos dará la posibilidad de hacer click sobre él y desencadenar un evento al que asociaremos un mecanismo para la selección de un valor perteneciente a la dimensión por la que queremos filtrar.

Al utilizar botones como elementos en las filas de la tabla, se dificulta la utilización de Drag&Drop para la eliminación de cada elemento seleccionado. Por esto, se ha añadido otra columna que para cada fila contendrá una imagen que al hacer click sobre ella se

eliminará dicha fila. Es decir, cada filtro seleccionado tendrá un doble mecanismo de eliminación, o bien arrastrando el elemento hasta la papelera, o haciendo click en la imagen que acompaña al botón de dicho elemento seleccionado.

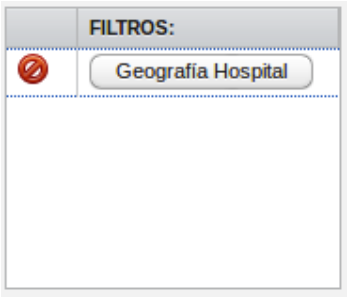


Figura 6.5: Dimensiones seleccionadas como filtro.

6.5.5. Selección del valor a filtrar

Una vez añadida una dimensión sobre la que queremos filtrar, es necesario dar la posibilidad de elegir los valores de esa dimensión que realmente van a actuar de filtro, es decir, que solo se mostrarán los datos consultados que tengan ese valor para esa dimensión.

Cada dimensión posee un conjunto de valores discretos por los que podemos filtrar, estos valores se pueden obtener consultando sobre el cubo. Al igual que ocurre cuando navegamos sobre el cubo, hay dimensiones que tienen gran cantidad de niveles de jerarquía, y tenemos que dar la posibilidad de filtrar por cualquiera de ellos. Para ello, ya hemos codificado anteriormente una función que nos devuelve los hijos de un elemento dado.

Necesitamos entonces una estructura que nos muestre los posibles valores jerarquizados disponibles para el filtrado y un listado con los ya seleccionados. Como ya hemos hecho anteriormente, elegiremos un árbol como estructura para la representación de la jerarquía de valores, y una tabla como componente para listar los valores seleccionados para un determinado filtro.

En este caso la tabla solo dispondrá de un mecanismo para eliminar los valores seleccionados anteriormente. Este mecanismo, ya visto anteriormente, constará de una columna adicional con una imagen que al actuar sobre ella eliminará la fila correspondiente.

El árbol mostrará inicialmente los valores de nivel superior, y los niveles inferiores serán cargados bajo demanda del usuario a través de nuevas consultas a través del módulo “Estado”.

Tanto el árbol como la tabla de valores seleccionados deben ser mostrados al actuar sobre el botón que es creado al seleccionar una dimensión como filtro en la ventana principal. Dicho botón creará una nueva ventana (diálogo modal) que contendrá la tabla y el árbol mencionados. La selección de uno de los valores del árbol se realizará también con el método

de Drag&Drop. La incorporación de un nuevo elemento a la tabla de valores seleccionados provocará una llamada al módulo de "Estado" y este se encargará de establecer el nuevo filtro. Tras incorporar o eliminar un valor filtrado, se rellenara la tabla de resultados con los nuevos datos consultados (y filtrados).

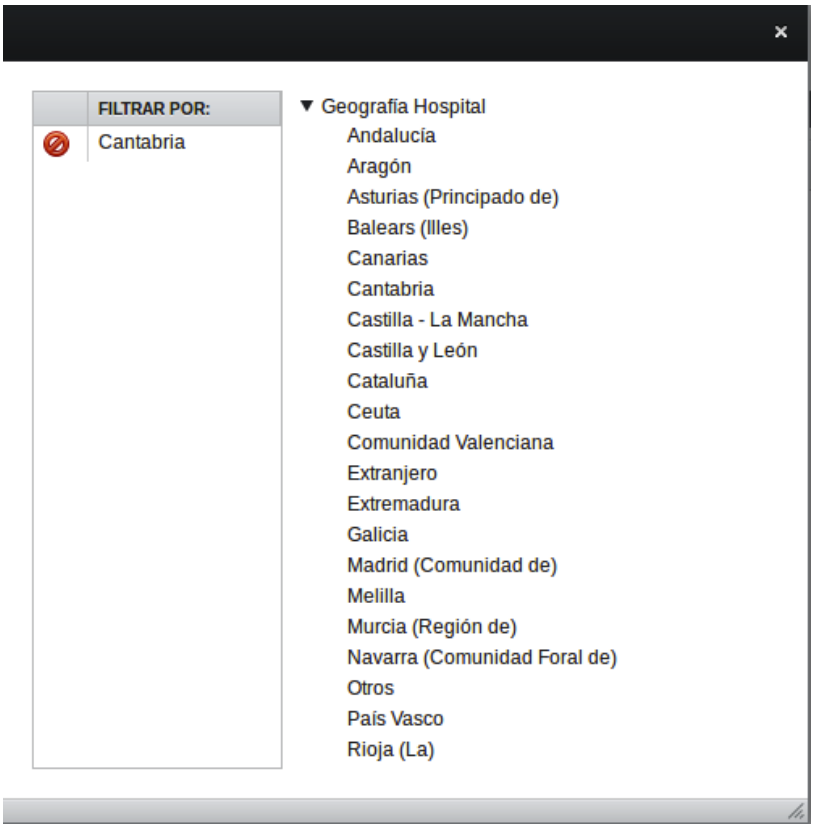


Figura 6.6: Posibles valores de una dimensión con los que filtrar.

6.5.6. Tabla multicabecera

Es un requisito preestablecido que los datos consultados sean representados de forma tabular, por lo tanto el componente visual asociado a los datos devueltos por la aplicación será una tabla.

El componente tabla de Vaadin ofrece una gran versatilidad. Nos permite añadir como elementos en las celdas una gran cantidad de componentes (botones, etiquetas, campos de texto...), dispone de una gran cantidad de eventos asociados, carga de datos bajo demanda, permite colapsar columnas, establecer un nivel de cabecera y pie de tabla, y muchas acciones más.

Sin embargo las características de la tabla que nos ofrece Vaadin como componente

no se ajusta a las necesidades requeridas por nuestra aplicación. Nuestros datos pueden ser representados con múltiples dimensiones tanto en las filas como en las columnas. Si tenemos más de una dimensión, o una dimensión y alguna métrica, seleccionadas en las columnas, los resultados aparecerán con varios niveles de cabeceras, cada uno correspondiente a una dimensión o métrica. El problema de la tabla por defecto es que solo permite un nivel de cabecera.

Esto ha supuesto un problema inesperado en la planificación puesto que ha sido necesario invertir tiempo en la modificación y desarrollo de una nueva tabla. Esta nueva tabla está basada en la anterior, pero se ha añadido el conjunto de características que cubren todas nuestras necesidades (multicabecera, ordenación de ciertas columnas y manejadores de eventos).

	MÉTRICAS	
	CASOS	ESTANCIA MEDIA
	SEXO	
TIPO INGRESO	+ TODOSLOSSEXOS	+ TODOSLOSSEXOS
+ Todos Los Ingresos	950.000	7,786

Items per page: 10 << < Page: 1 / 1 > >>

Figura 6.7: Tabla con múltiple cabecera

6.5.7. Gráfica

La representación de los datos en forma gráfica es también otro requisito. Para ello se ha creado un componente que asociado a la librería de contenido gráfico jFreeChart [9] es capaz de mostrar una gráfica como una imagen en nuestra aplicación.

Este componente forma parte del módulo gráfico que permite además de la creación de la gráfica, la actualización de la misma. Este mecanismo de actualización de datos será utilizado por el estado para reflejar los nuevos datos que han sido consultados. Para realizar la actualización es necesario indicar al módulo gráfico todos los datos relacionados con los

datos consultados, siguiendo en este caso la estructura que hemos definido anteriormente, es decir, cabeceras de las columnas, cabeceras de las filas y valores.

Además de actualizar los datos también es posible configurar algunos parámetros como el tipo de gráfico mostrado (porciones o barras) o el tamaño del mismo.

Se ha decidido que las dos posibles vistas de representación de datos anteriores (tabular o gráfica), no sean visibles simultáneamente, si no que estén incluidas dentro de un panel con dos pestañas que permita elegir una vista u otra.

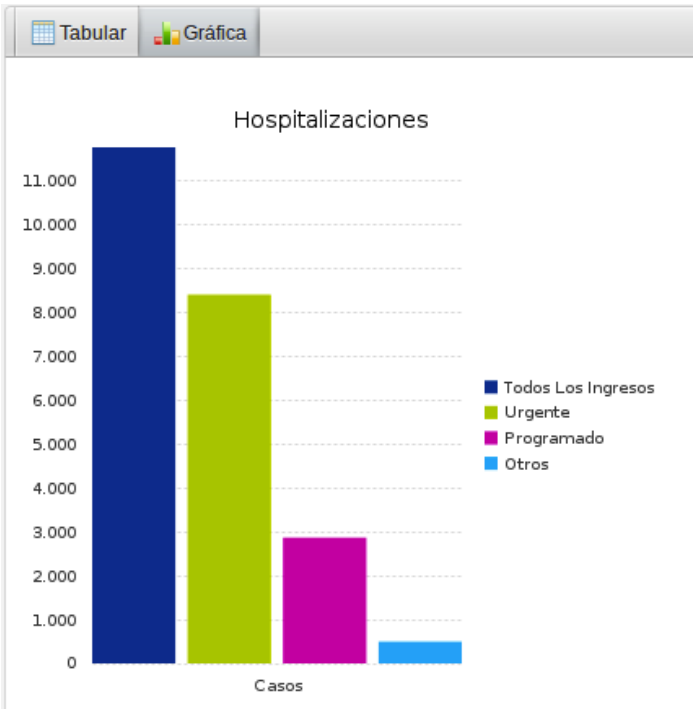


Figura 6.8: Visualización de los datos en forma gráfica.

6.5.8. Traducción de textos y mensajes (i18n)

Uno de los requisitos es dar la posibilidad de internacionalizar la aplicación, es decir, poder personalizar los textos que se muestran tanto en menús como en mensajes.

Para realizar esto, se ha creado una clase “TraduccionTextos” en la que se definen dos métodos por cada texto de la aplicación, uno para obtener el actual y otro para establecer un nuevo contenido para ese texto. Como los textos son asignados en la inicialización de la interfaz, es necesario que los cambios deseados se realicen antes de la llamada al constructor de la interfaz y se le pase como parámetro una instancia de la clase mencionada con los textos deseados ya establecidos.

6.5.9. Menú opciones

Algunos elementos tienen ciertos comportamientos que el usuario puede modificar interactuando con la interfaz. Por esto se han creado diferentes conjuntos de opciones a través de componentes de barras de menú.A continuación se definen los diferentes menús y sus opciones:

- Opciones generales:
  - Exportar a XLS: permite exportar el contenido de la tabla.
  - Reiniciar conexión OLAP: realiza un reinicio del enlace con el servidor de cubos.
- Opciones tabla:
  - Ocultar filas con todos los valores vacíos: Al realizar una consulta se omite la visualización de aquellas filas que tienen todos sus valores vacíos.
  - Pivotar: Es la opción que permite intercambiar el contenido de los elementos seleccionados en filas y columnas.
- Opciones gráfica:
  - Ocultar valores acumulados en filas expandidas: Al representar los datos gráficamente omite los valores de las filas sumarizadas.
  - Mostrar solo máximo nivel de detalle: Representa gráficamente solo aquellas filas que tienen una profundidad mayor, es decir, los últimos elementos obtenidos con la última navegación drill-down.
  - Actualizar gráfica automáticamente: Esta opción permite que se actualicen los datos de la gráfica mientras se modifica el contenido de las filas o columnas seleccionadas, filtros, o algunas de las opciones anteriores.
  - Tamaño: Permite establecer el tamaño del gráfico mostrado de entre una lista de posibles valores. Las unidades se encuentran en pixels.
  - Tipo: permite elegir la tipología del gráfico mostrado. Hay dos opciones disponibles: gráfico de barras o de porciones.

6.6. Traducción de elementos consultados

Dado que en las bases de datos, o en los Data Marts, muchas veces la información no se encuentra almacenada de forma explícita o entendible directamente, es necesario un mecanismo de transformación (traducción) para ciertos valores.

Al codificar el test de este requisito definimos primero un objeto que va a actuar de traductor, en el que a través de un método establecemos una correspondencia entre el elemento “Sexo.1” y “Masculino”. A continuación vinculamos ese traductor a un “Estado”

en el que hemos seleccionado “Sexo” en filas y “Casos” en columnas. Navegamos sobre ese estado y comprobamos que en los datos devueltos aparece la celda “Masculino” en lugar de “1”.

Esta prueba nos lleva a codificar varios elementos. Primero tenemos que definir la estructura del traductor, para que tanto el usuario, que es el que envía el elemento, como la aplicación, que es la que lo aplica, entiendan el objeto. Para ello creamos la interfaz “ITraductorDeValores” que define que el objeto ha de tener un método para establecer una traducción (asociar un valor original a otro nuevo), un método de consulta para saber si un valor tiene asociada una traducción y por último un método para consultar la traducción de un valor.

Una vez definido el objeto traductor creamos un método en el módulo “Estado” para poder asignar un objeto de tipo “ITraductorDeValores” a un estado. Este método será público y se podrá invocar a través del componente creado por una aplicación Vaadin, de esta manera se puede variar la traducción de forma dinámica y no necesariamente al crear el componente.

La traducción tiene que aplicarse entre la etapa de recepción de datos de Mondrian y la de presentación al usuario. Aprovechamos entonces el método que se dedica a distribuir los datos originales en las diferentes estructuras definidas en 6.2. Esta traducción, sin embargo, solo afecta al texto mostrado gracias a que los objetos “ICelda” pueden diferenciar entre lo que se muestra y el contenido real o el ID de cada elemento. Estos textos traducidos solo serán visibles en la tabla de resultados y en la lista de valores a filtrar.

6.7. Vista global

Por último y a modo de resumen del proceso de diseño y construcción llevado acabo en este capítulo, se muestra en la Figura 6.9 con la estructura de clases que ha resultado.

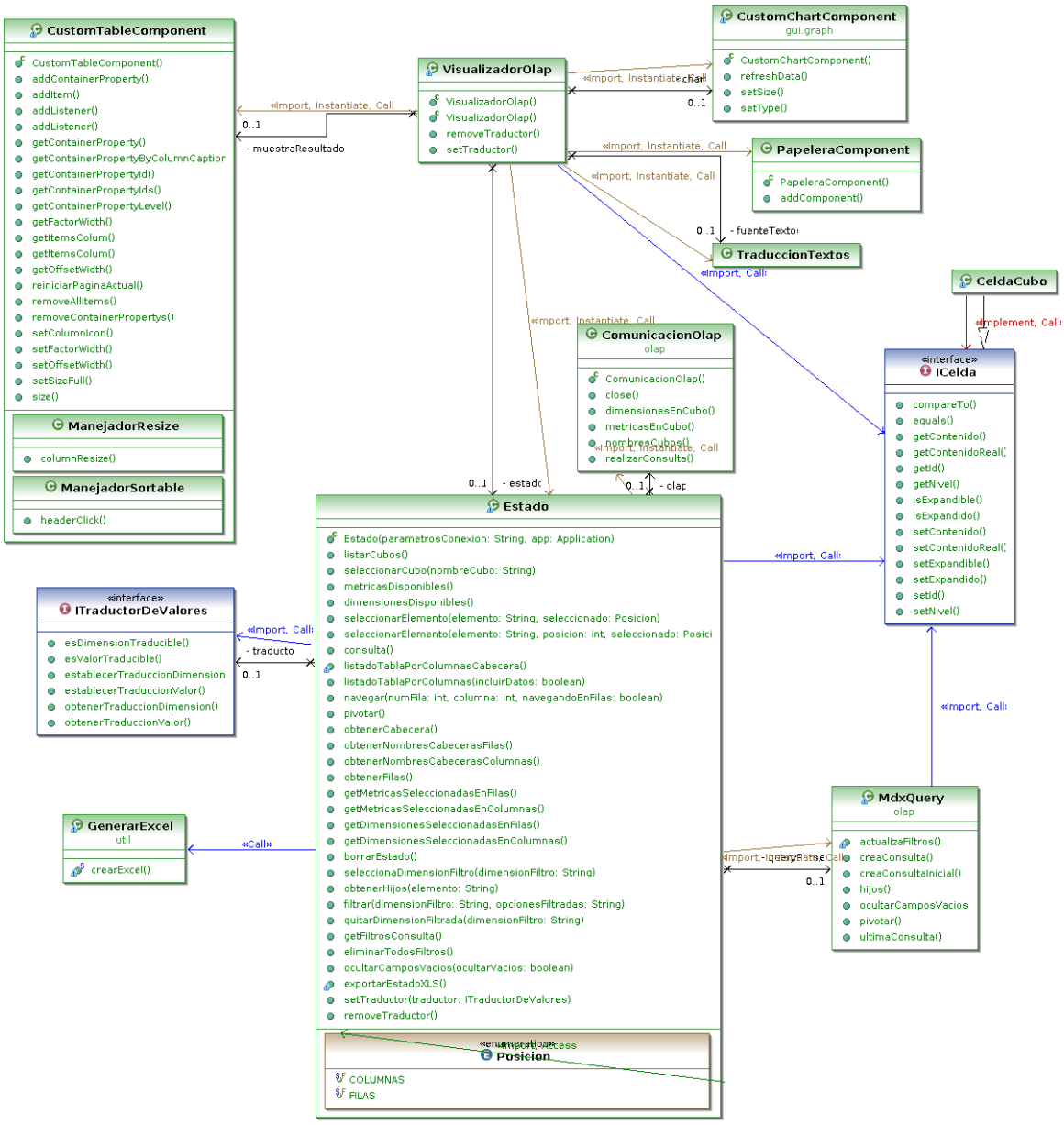


Figura 6.9: Estructura de clases del componente desarrollado

Del mismo modo, tras el proceso de diseño y construcción de la interfaz gráfica, se muestra en la Figura 6.10 el resultado final con todos los elementos organizados y distribuidos de forma apropiada.

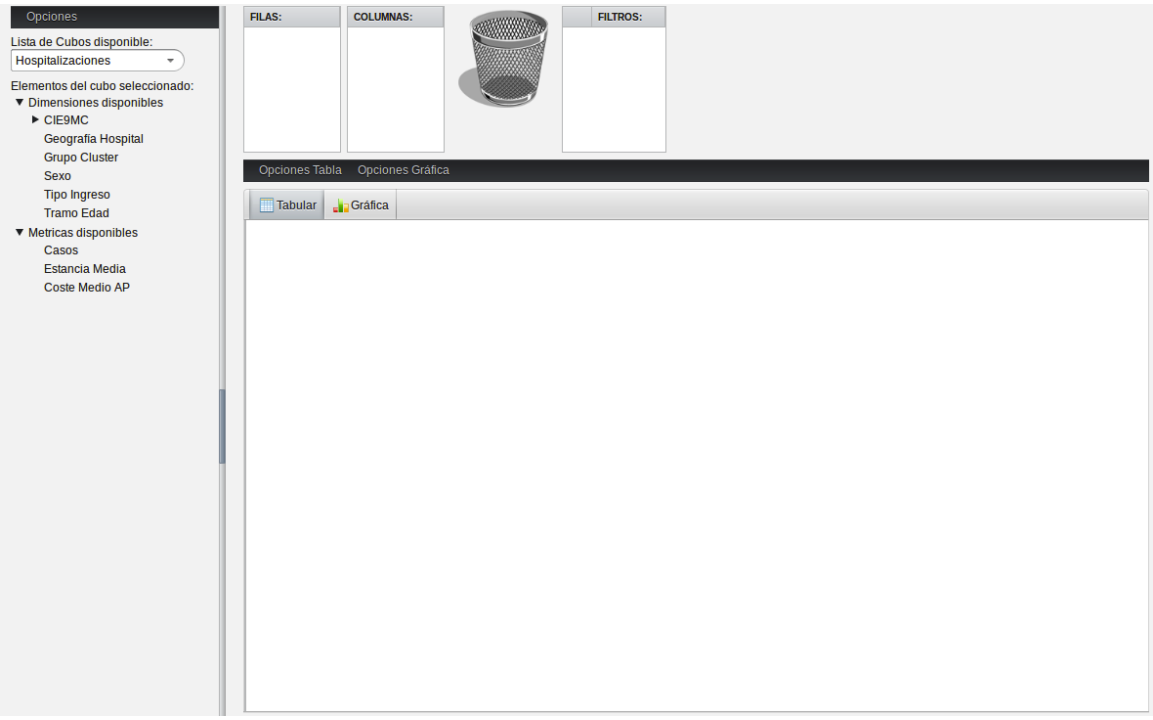


Figura 6.10: Apariencia visual de la interfaz de la aplicación

Capítulo 7

Conclusiones y Trabajos Futuros

Como parte final de la memoria, se muestra las conclusiones del proyecto así como posibles proyectos futuros.

Índice

7.1. Conclusiones

Los objetivos de este Proyecto de Fin de Carrera, descritos en la seccion 2.1, se han cumplido en su totalidad. Se comenzó realizando un aprendizaje sobre el uso de la herramienta de desarrollo Vaadin y el manejo y comportamiento del servidor de cubos Mondrian. Tras la fase de aprendizaje se realizó un estudio sobre aplicaciones de similares características a la deseada. Por último se diseñaron y codificaron las pruebas y el código de la aplicación.

Se ha conseguido desarrollar un componente reutilizable y fácil de usar, capaz de navegar, extraer y mostrar la información proporcionada por un cubo OLAP. Este componente ha sido integrado y probado en una aplicación real de la empresa en la que se ha realizado este proyecto.

Se han obtenido gran cantidad de conocimientos durante su realización, además de afianzar los estudiados durante la carrera en diversas asignaturas (bases de datos, Programación, Ingeniería del Software, Sistemas de Información en la Empresa, etc). A destacar la importancia que ha tenido plasmar lo aprendido en el mundo real, porque en esta ocasión se ha desarrollado una aplicación que tenía unos requisitos muy específicos y que se espera que cubra una necesidad.

Acercándonos un poco más a la realidad, durante el transcurso de este proyecto han existido retrasos frente a las estimaciones de tiempo realizadas inicialmente. Se realizó una

aproximación por la que cada requisito debía llevar una semana de trabajo y a este tiempo habría que añadirle un mes para la fase de aprendizaje de nuevas tecnologías. Sin embargo durante el desarrollo del proceso surgieron retrasos debidos a la necesidad de modificar el funcionamiento de la tabla por defecto que contiene Vaadin. Este trabajo ha requerido de casi un mes más de tiempo para la realización total de este proyecto.

7.2. Trabajos Futuros

En un futuro próximo, una vez finalizados los trámites que se están llevando a cabo, se pondrá a disposición de todo el mundo, en la pagina de addons de Vaadin [15], el componente desarrollado así como el código fuente para que pueda ser modificado, mejorado o ampliado. Del mismo modo ocurrirá con la tabla desarrollada, es un componente al que se le ha añadido una funcionalidad y será puesto al alcance de toda la comunidad.

A medio plazo sería necesario añadir un nuevo sistema de representación gráfica de los datos, puesto que el actual tiene algunas limitaciones en cuanto a personalización y representación de etiquetas de texto demasiado grandes. Una buena alternativa consistiría en gráficas mostradas mediante HTML5, que darían además la posibilidad de ser interactivas. Otra posible ampliación consistiría en añadir una opción de exportación a formato pdf, tanto para los datos en forma tabular como para las gráficas.

A largo plazo una buena mejora consistiría en permitir la reorganización manual de los distintos elementos de la interfaz por parte del usuario, para personalizar el estilo y poder conseguir así una integración completa del componente con el resto de la aplicación.

Apéndice A

Contenidos del CD

En este capítulo se muestra el contenido del CD adjunto a esta memoria, el cual aporta contenido multimedia así como una copia digital de esta memoria.

El CD contiene los siguientes elementos:

- **Memoria del proyecto:** Memoria del proyecto en formato *pdf*.
- **VideoTutoriales:** Videotutoriales con las acciones más comunes de la aplicación.

Bibliografía

[1] *Introducción a Pentaho Business Intelligence.* [Consulta: 26 de Marzo de 2012].
http://pentaho.almacen-datos.com/

[2] *Pentaho Business Analytics - Business Intelligence, Data Integration and Data Mining Solution - Pentaho.* [Consulta: 26 de Marzo de 2012].
http://www.pentaho.com/explore/pentaho-business-analytics/

[3] Enric Biosca. *Tutorial MDX.* [Consulta: 28 de Marzo de 2012].
http://www.dataprix.com/files/Tutorial_MDX.pdf

[4] Josep Lluís Cano. *BUSINESS INTELLIGENCE: COMPETIR CON INFORMACIÓN.* [Consulta: 28 de Marzo de 2012].
http://www.iwith.org/pdf/Libro_BI_Competir_con_Informacion.pdf

[5] Ministerio de sanidad, servicios sociales e igualdad. *Consulta Interactiva del SNS.* [Consulta: 17 de Mayo de 2012].
http://pestadistico.msc.es/PEMSC25/ArbolNodos.aspx

[6] Ministerio de sanidad, servicios sociales e igualdad. *Portal Estadístico del SNS.* [Consulta: 17 de Mayo de 2012].
http://www.msc.es/estadEstudios/estadisticas/cmbd.htm

[7] IASIST. *Conjunto Mínimo Básico de Datos (CMBD).* [Consulta: 17 de Mayo de 2012].
http://www.iasist.com/es/recursos/glosario/conjunto-minimo-basico-de-datos-cmbd

[8] Apache Software Foundation. *The Java API for Microsoft Documents.* [Consulta: 4 de Junio de 2012]
http://poi.apache.org/index.html

[9] Object Refinery Limited. *JFreeChart.* [Consulta: 28 de Mayo de 2012].
http://www.jfree.org/jfreechart/

[10] *JPivot.* [Consulta: 22 de Febrero de 2012].
http://jpivot.sourceforge.net/

[11] Mondrian lead developer. *Pentaho Analyzer.*[Consulta: 1 de Julio de 2012].
http://forums.pentaho.com/showthread.php?72691-Pentaho-Analyzer

[12] Apache Software Foundation. *Maven.* [Consulta: 1 de Julio de 2012].
http://maven.apache.org/

[13] Vaadin Ltd. *Introducción al desarrollo en Vaadin.* [Consulta: 6 de Febrero de 2012].
https://vaadin.com/learn

[14] Dundas Data Visualization. *Dundas Chart for .NET OLAP Services.* [Consulta: 26 de Marzo de 2012].
http://demos3.dundas.com/olapdemostaging55

[15] Vaadin Ltd. *Directorio de Add-ons de Vaadin.* [Consulta: 6 de Febrero de 2012].
https://vaadin.com/directory

[16] Dundas Data Visualization. *Sobre Dundas Data Visualization.* [Consulta: 26 de Marzo de 2012].
http://www.dundas.com/corporate/

[17] Jim Highsmith. *Manifiesto sobre metodologías ágiles.* [Consulta: 5 de Junio de 2012].
http://agilemanifesto.org/history.html

[18] Jorge Rubira. *Solid, cinco principio básicos de diseño de clases.* [Consulta: 28 de Mayo de 2012].
http://www.genbetadev.com/paradigmas-de-programacion/solid-cinco-principios-basicos-de-diseno-de-clases

[19] The Eclipse Foundation. *Sobre Eclipse Foundation.* [Consulta: 6 de Febrero de 2012].
http://www.eclipse.org/org/

[20] *Comunidad jUnit.* [Consulta: 12 de Marzo de 2012].
http://www.junit.org

[21] Google. *Google Web Toolkit.* [Consulta: 12 de Marzo de 2012].
https://developers.google.com/web-toolkit/overview

[pent] Roland Bouman, Jos van Dongen,
Pentaho solutions : business intelligence and data warehousing with Pentaho and MySQL, Indianapolis, 2009

[design] Kenneth E. Kendall,Julie E. Kendall,
Análisis y Diseño de Sistemas, Prentice Hall, 2005

[junit1] Kent Beck,
JUnit Pocket Guide, O'REILLY & ASSOCIATES, 2004

[etl1] Ralph Kimball, Joe Caserta,
The Data Warehouse ETL Toolkit: Practical Techniques for Extractin, Cleaning, Conforming, and Delivering Data, WILEY, versión Ebook 2011