Aplicación para el cálculo de consumos energéticos en el CERN, empleando el histórico del sistema de control eléctrico (SCADA)

Máster Universitario en Investigación en Ingeniería del Software y Sistemas Informáticos

Universidad Nacional de Educación a Distancia

Septiembre 2015, Curso 2014-2015

Autor: Alejandro Camargo Barrantes

Director: Rubén Heradio Gil

Aplicación para el cálculo de consumos energéticos en el CERN, empleando el histórico del sistema de control eléctrico (SCADA)

Máster Universitario en Investigación en Ingeniería del Software y Sistemas Informáticos

Universidad Nacional de Educación a Distancia

Septiembre 2015, Curso 2014-2015

Autor: Alejandro Camargo Barrantes

Director: Rubén Heradio Gil

Trabajo específico propuesto por el alumno

Código: 31105128

Espacio reservado para la hoja de calificaciones

Autorización

Autorizo a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del autor

Agradecimientos

Gracias a Rubén Heradio, Sonia Infante y Anargyros Kiourkos por su apoyo técnico.

Y gracias a Beatriz.

Resumen

Este documento tratará de explicar el Proyecto *WebEnergy*, desarrollado en la sección EL-CO en el CERN, por el autor.

El objetivo del Proyecto es, empleando la base de datos del histórico del sistema SCADA, calcular los consumos eléctricos en diferentes sitios de interés en el CERN como puedan ser edificios, experimentos, complejos, etc., y mostrarlos en gráficas en una aplicación web, de manera que puedan ser monitorizados.

Antes de *WebEnergy* no existía un Sistema centralizado para calcular estos consumos y el consumo global del CERN venía dado directamente por el proveedor eléctrico, ya fuera el suizo o el francés.

La importancia de *WebEnergy* no es solo proveer al CERN de una herramienta que proporcione los consumos en las diferentes instalaciones (que podrá ser de utilidad para los físicos e ingenieros) sino también poder saber con antelación el consumo que vendrá en la próxima factura eléctrica (y que será útil para los departamentos de finanzas).

El autor ha puesto en práctica los conocimientos adquiridos en el máster para desarrollar una arquitectura sostenible, aplicando principios de desarrollo, patrones de diseño y creando una arquitectura orientada a servicios que permitirá conectar *WebEnergy* con otras aplicaciones en el CERN gracias a los webservices.

También, se han empleado estrategias de generación automática de código para generar las capas de persistencia, reduciendo así la cantidad de código repetitivo a escribir.

En las próximas páginas se describirán los procedimientos, arquitectura y diseño que se han realizado en este Proyecto, esperando que la experiencia reunida en el desarrollo de *WebEnergy*, pueda inspirar a otras organizaciones con necesidades parecidas, a que implementen una solución similar.

WebEnergy consiguió de manera exitosa calcular los consumos del CERN con un error en torno al ~0.7%.

Abstract

This document will try to explain the *WebEnergy* project, developed within the EL-CO section at CERN by the author.

The aim of the project is, using the SCADA system archive database, to calculate energy consumptions in different interesting places at CERN, like experiments, buildings, facilities, etc., and show them in trends in a web application, so they can be monitored.

Before *WebEnergy*, there was no centralised system to calculate the consumptions in these places. CERN global consumption was given directly by the energy supplier, either French or Swiss.

The importance of *WebEnergy* is not only to provide CERN a tool that will tell the energy consumed by the different facilities (useful for the physics and engineering), but to know in advance the amount of the next electrical invoice (useful for the finance departments).

The author has put in practice the knowledge accumulated during the master degree for designing a sustainable architecture, applying development principles, design patterns and creating a service oriented architecture to interconnect *WebEnergy* with other CERN applications thanks to webservices.

Also, code generation strategies have been applied to generate the persistence layers, reducing the amount of repetitive coding work.

So, in the next pages, the application's different procedures, design and architecture will be described hoping that the experience gathered in the development of this project, could inspire other organisations with similar needs and characteristics to implement a similar solution.

The *WebEnergy* project successfully managed to calculate the CERN energy consumptions with an error around ~0.7%.

Keywords

CERN, Energy consumption, Software Oriented Architecture, Automatic Code Generation, Software Principles, Design Patterns, Software Architecture, Software Reuse.

Índice

1 Introducción 16

2 Objetivos 18

2.1 Contexto 18

2.1.1 Introducción 18

2.1.2 El CERN 18

2.1.3 Las subestaciones eléctricas y el sistema SCADA 19

2.1.4 Los datapoints 21

2.1.5 La base de datos del histórico 21

2.1.6 La base de datos LHC Logging 22

2.1.7 Conclusiones 22

2.2 Requisitos funcionales 22

2.3 Requisitos no funcionales 23

3 Solución desarrollada 25

3.1 Fundamentos 25

3.1.1 Introducción 25

3.1.2 Una visión global 25

3.1.3 Estructurando los datos 26

3.1.3.1 Datapoints 26

3.1.3.2 Feeders 27

3.1.3.3 Consumers 27

3.1.3.4 Super Consumers 28

3.1.4 El modelo de datos 29

3.1.4.1 Representación 29

3.1.4.2 Tabla WEB_DATAPOINT 30

3.1.4.3 Tabla WEB_FEEDER 31

3.1.4.4 Tabla WEB_CONSUMER 31

3.1.5 El cálculo de los consumos 32

3.1.5.1 Importación y cálculo de los valores diarios de los datapoints 32

3.1.5.2 Posibles errores 33

3.1.5.3 Cálculo de los valores de los Feeders 34
3.1.5.4 Cálculo del árbol de Consumers. 35
3.1.6 Interacción entre las distintas aplicaciones 38
3.2 Arquitectura de la aplicación 39
3.2.1 Introducción 39
3.2.2 Principios arquitectónicos de la aplicación 39
3.2.2.1 La capa de datos 40
3.2.2.2 La capa de negocio y la arquitectura orientada a servicios (SOA) 41
3.2.2.2.1 Capa de servicios de acceso a datos 42
3.2.2.2.2 Capa de servicios de negocio 43
3.2.2.2.3 Capa de servicios de integración 43
3.2.2.2.4 La capa de negocio y Spring 44
3.2.2.3 La capa de presentación 44
3.2.3 Seguridad 45
3.3 Diseño de la aplicación para facilitar la mantenibilidad 45
3.3.1 Introducción 45
3.3.2 Los principios ISP y el OCP 46
3.3.3 El principio DRY y los patrones de diseño DAO y GenericDAO 46
3.3.4 La capa de servicios de acceso a datos 49
3.3.5 La capa de presentación y Java Server Faces 50
3.3.6 La inversión del control (IoC) 51
3.3.7 Generación automática del código de las capas de persistencia y servicios 51
3.3.7.1 Introducción 51
3.3.7.2 Identificación de la línea de productos 53
3.3.7.3 Definición de un lenguaje específico de dominio (DSL) 54
3.3.7.4 El generador de código 55
3.3.7.5 Conclusiones 56
4 Conclusiones y trabajo futuro 57
4.1 Resultados 57
4.1.1 Introducción 57
4.1.2 Consumo real vs. Consumo estimado por WebEnergy 57
4.1.3 Error medio y desviación típica 60

4.2 Trabajo futuro 61
4.2.1 Introducción 61
4.2.2 Aumento de la resolución 61
4.2.3 Creación de nuevos árboles 61
4.2.4 Mapa de energía 62
4.2.5 Ampliación de la API REST para publicación de datos 62
4.2.6 Cálculo de la potencia 62
4.2.7 Exportación a otros entornos industriales 63
5 Referencias 64
Anexo: tabla de consumos 65

Índice de figuras

Figura 1: Mapa del CERN 19
Figura 2: Sistema SCADA de la red eléctrica 20
Figura 3: Flujo de datos 26
Figura 4: Representación de las diferentes entidades 29
Figura 5: Modelo de datos 30
Figura 6: Aspecto de la tabla WEB_DATAPOINT 30
Figura 7: Aspecto de la tabla WEB_FEEDER 31
Figura 8: Aspecto de la tabla WEB_CONSUMER 31
Figura 9: Valores para EKD204/5E_Ea+ 32
Figura 10: Valores para EKD204/5E_Ea- 33
Figura 11: Ejemplo de reseteo de contador 34
Figura 12: Valores para el feeder EKD204/5E 35
Figura 13: Recorrido del árbol 36
Figura 14: Aspecto de la tabla CEM_E_P_DAILY 38
Figura 15: Modelo por capas 40
Figura 16: Aplicación orientada a servicios 42
Figura 17: El patrón DAO 47
Figura 18: Programación contra interfaces 47
Figura 19: Patrón Generic DAO 48
Figura 20: Capa de servicios de acceso a datos 49
Figura 21: Modelo-Vista-Controlador 50
Figura 22: El generador de código 55
Figura 23: Consumo real vs. Consumo estimado 59
Figura 24: Error en % del consumo estimado respecto al valor real 60

Índice de listados

Listado 1: Cálculo del consumo para un Consumer 37
Listado 2: Cálculo del consumo de un Super Consumer 37
Listado 3: Ejemplo de interfaz que hereda del DAO Genérico 54
Listado 4: Ejemplo de DSL 54

Lista de símbolos

ALICE	*A Large Ion Collider Experiment*
API	*Application Programming Interface*
ATLAS	*A Toroidal LHC Apparatus*
CCC	*CERN Control Centre*
CERN	*Organisation Européene pour la Recherche Nucléaire*
CMS	*Compact Muon Solenoid*
CO	*Controls Section*
CSV	*Comma Separated Values*
DAO	*Data Access Object*
DI	*Dependency Injection*
DRY	*Don't Repeat Yourself*
DSL	*Domain Specific Language*
EL	*Electrical Group*
EN	*Engineering Department*
GP	*Generative Programming*
HQL	*Hibernate Query Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IoC	*Inversion of Control*
ISP	*Interface Segregation Principle*
JEE	*Java Enterprise Edition*
JPA	*Java Persistence API*
JSF	*Java Server Faces*

JSON	*Javascript Object Notation*
JSP	*Java Server Pages*
kWh	*Kilo Watios Hora*
LHC	*Large Hadron Collider*
LHCb	*Large Hadron Collider Beautiful Experiment*
OCP	*Open Closed Principle*
ORM	*Object-Relational Mapping*
POJO	*Plain Old Java Object*
PS	*Proton Synchrotron*
REST	*Representational State Transfer*
RTU	*Remote Unit*
SCADA	*Supervisory Control And Data Acquisition*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SPS	*Super Proton Synchrotron*
SQL	*Standard Query Language*
XML	*eXtensible Markup Language*

1 Introducción

Este proyecto se ubica dentro del CERN, en el departamento de ingeniería (EN) y dentro la sección control eléctrico[1] (EL-CO). Se ha encomendado al autor el desarrollo del mismo para responder a la necesidad de contar con una herramienta que permitiera monitorizar los consumos de energía que se producen diariamente en los diversos experimentos, edificios y lugares de interés en el CERN. Hasta ese momento no se tenía una herramienta similar, contándose con tan solo la factura final del proveedor de electricidad Francés (el principal) o Suizo (secundario o backup).

La creación de esta herramienta –bautizada como *WebEnergy*– pretende a *grosso modo*, emplear los datos del histórico del sistema de control eléctrico SCADA (basado en WinCC OA de Siemens) albergados en una base de datos, para realizar el cálculo de los consumos de energía y mostrarlos en una aplicación web que permita su visualización. De esta manera se tendrán los datos de consumos diarios y una serie de gráficos que facilitarán su seguimiento.

Este cálculo se realizará diariamente, obteniendo los valores de consumos para los distintos elementos que se desean estudiar. Gracias a los resultados obtenidos se podrá tener esta información sobre los consumos de diversos elementos importantes que de otra manera no se tendría (consumos para el LHC, SPS o para el Centro de Cálculo entre otros). De alguna manera, se podrán segregar los consumos obteniendo una información de la que antes se carecía.

Esto no solo permitirá obtener datos valiosos para la física y los distintos experimentos, sino que también permitirá al departamento de finanzas adelantar la factura de la electricidad y realizar el seguimiento de la energía consumida en el transcurso de los días.

El autor se ha servido de los conocimientos adquiridos en el máster para diseñar la arquitectura y desarrollar WebEnergy, aplicando principios de desarrollo, patrones de diseño y creando una aplicación orientada a servicios que ha

[1] http://en.web.cern.ch/en-el-group

facilitado la reutilización sistemática de código, así como la interconexión con otras aplicaciones dentro del CERN mediante *webservices*.

También se han empleado técnicas de generación automática de código para generar la capa de persistencia y de servicios de acceso a datos, consiguiendo así la reducción de la realización de tareas de codificación repetitivas.

La experiencia acumulada en la realización de este proyecto puede servir de inspiración para otras organizaciones o compañías que contando con un sistema SCADA puedan aplicar una solución similar a la aquí presentada para calcular sus consumos energéticos con antelación a la factura de luz de su proveedor.

Este documento se ha estructurado como sigue: (i) en el capítulo de Objetivos se aborda el contexto en el que se desarrolla el proyecto y los requisitos funcionales y no funcionales. (ii) En el capítulo Solución Desarrollada, se profundiza en los fundamentos teóricos del proyecto y su entorno así como se explica la arquitectura y el diseño de la aplicación desarrollada, incluyendo las estrategias empleadas para garantizar la mantenibilidad del código. (iii) Finalmente en el capítulo de Conclusiones y Trabajo Futuro, se exponen los resultados del proyecto, realizándose la comparación de consumos reales y estimados, así como las posibles expansiones para el proyecto.

2 Objetivos

2.1 Contexto

2.1.1 Introducción

En este apartado se procederá a explicar el entorno en el que se desarrolla el proyecto de modo que sea más comprensible ubicar los objetivos del desarrollo así como la justificación de las decisiones de diseño y arquitectura que se han tomado en el mismo.

2.1.2 El CERN

El CERN es un organismo internacional dedicado a la investigación que se sitúa en la frontera franco-suiza. Quizás es más conocido porque entre sus experimentos se encuentra el famoso LHC (*Large Hadron Collider*), el acelerador de partículas más grande del mundo.

En la Figura 1 se puede observar el mapa del CERN, que cuenta con dos *sitios* principales: el sitio de Meyrin (el más grande e importante) y el sitio de Prevessin. Ambos sitios se componen de edificios con oficinas, talleres, almacenes, experimentos etc., siendo similares a polígonos industriales.

En el subsuelo se encuentran los experimentos más importantes, entre ellos el LHC: un túnel de 27 km donde se encuentra el acelerador de partículas y el SPS (*Super Proton Synchrotron*), antiguo acelerador que ha sido reciclado como etapa intermedia para acelerar el haz de partículas que será inyectado posteriormente en el LHC.

A lo largo del perímetro del LHC en la superficie, se encuentran varios puntos numerados del uno al ocho. Estos puntos contienen también instalaciones que sirven entre otras cosas para acceder a túnel mediante ascensores.

En algunos de estos puntos en el subsuelo se encuentran cavernas que albergan los grandes detectores (los lugares donde se producirán las colisiones en el acelerador). Estos son el Punto 1: con el detector *ATLAS*, el Punto 2 con *ALICE*, el Punto 5 con el *CMS* y el Punto 8 con el *LHCb*.

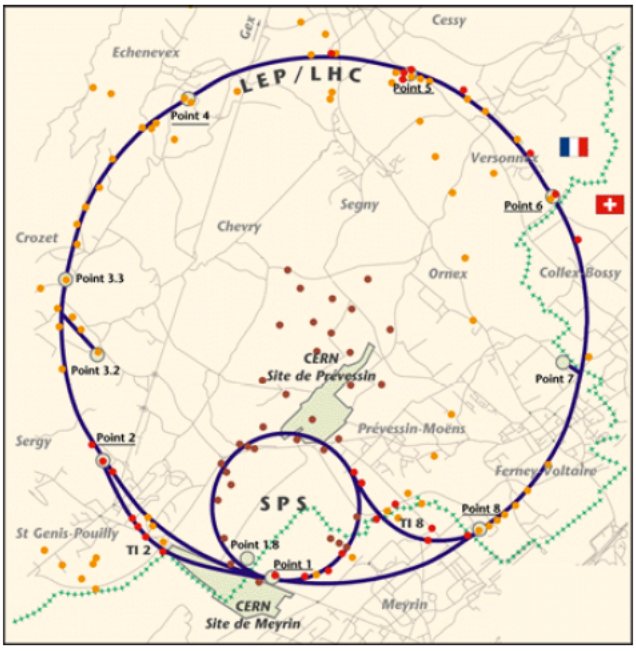

Figura 1: Mapa del CERN

En cada uno de los puntos, así como en los sitios de Meyrin y Prevessin y en el subsuelo (en las cavernas y en numerosos lugares dentro del túnel) se encuentran subestaciones eléctricas que proporcionan la energía necesaria a las instalaciones y experimentos.

2.1.3 Las subestaciones eléctricas y el sistema SCADA

Existen numerosas subestaciones eléctricas en el CERN. Es en las subestaciones eléctricas donde se encuentran los equipos necesarios para suministrar la energía (baterías, diéseles de emergencia, breakers…). Estos equipos son supervisados por una serie de ordenadores que se comunican con los mismos mediante una serie de conexiones y protocolos que van desde ethernet a puertos serie. Estos ordenadores recogerán información del estado de estos equipos, mediante una serie de *datapoints* que son definidos en la base de datos del sistema SCADA.

Cada uno de estos datapoints alberga una medición –un valor– que proporciona el equipo dado. Muy a grosso modo, se tendrán definidos unos valores límite para estos datapoints, que en el caso de ser sobrepasados podrán generar una alarma.

Los ordenadores (llamados *RTU, remote units*) enviarán las lecturas de estos datapoints a un sistema SCADA central bajo WinCC OA. En el caso de que los valores sean anómalos y generen una alarma, los operadores de la sala de control (CCC, *Cern Control Centre*) verán la alarma en su pantalla y llamarán a los responsables para que el problema pueda ser resuelto.

Un sistema SCADA es un sistema de control que permite la supervisión en tiempo real en un entorno industrial. En este caso supervisará el estado de la red eléctrica y de los equipamientos que existen.

En la Figura 2 se puede ver un esquema muy simplificado de cómo funciona este sistema en el entorno del proyecto. En cada una de las subestaciones eléctricas se tiene una RTU que se comunica con los equipos eléctricos y proporciona información en tiempo real a un servidor central.

Los operadores de la sala de control monitorizarán el estado de la red eléctrica (24x7) mediante un panel de alarmas que aparecerán cuando algún valor de la lectura de los datapoints sea anómalo y actuarán en consecuencia.

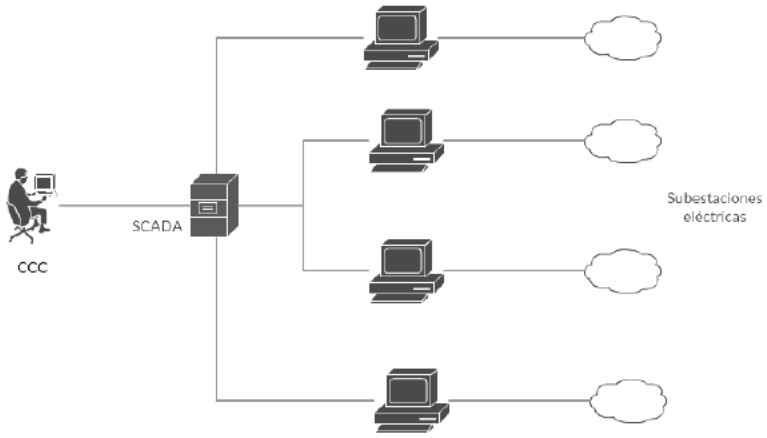

Figura 2: Sistema SCADA de la red eléctrica

Por supuesto, la red eléctrica está en continuo cambio y el sistema SCADA debe ser actualizado en consecuencia. Frecuentemente nuevos equipos son añadidos, sustituidos o retirados de la red y el sistema debe ser modificado en consecuencia para garantizar su fiabilidad.

2.1.4 Los datapoints

Los datapoints albergan mediciones de diversos parámetros de los equipos en las subestaciones eléctricas. Pueden representar valores tales como el voltaje, la temperatura, un estado (booleano) etc.

Estos valores son proporcionados por los equipos a las RTU en tiempo real mediante Ethernet, puerto serie, lonworks, etc. Dependiendo del equipo también lo harán empleando diferentes protocolos como IEC61850.

La mayor parte de los datapoints son irrelevantes para el propósito de este texto, excepto aquellos relativos a la energía. Existen una serie de datapoints que actúan a modo de contadores de energía en distintos equipos que son los que empleará la aplicación para realizar los cálculos.

Estos datapoints tendrán los sufijos *Ea+* y *Ea-* y representarán la energía entrante y saliente que circula por un determinado equipo en kWh. Con una sencilla resta *Ea+* menos *Ea-* se obtendrá el consumo de energía medido por el dispositivo.

2.1.5 La base de datos del histórico

El sistema SCADA contará con una base de datos con el histórico de los valores que se miden en los distintos datapoints. Esto es vital para operaciones de *post-mortem análisis.*

La aplicación recuperará los datos del histórico para los datapoints relacionados con la energía, pero ¿cómo y cuándo se almacenan esos valores en el histórico?

Como se ha comentado, la medición de los datapoints es continua y en tiempo real. Esto significa que los valores se generan continuamente. Cuando no exista consumo en algunos de estos datapoints, el valor no cambiará. Esto puede ser incluso durante largos periodos de tiempo, por lo que no tiene sentido almacenar todos los valores continuamente (lo que llevaría a hacer crecer la base de datos rápidamente con el consiguiente problema de espacio y sobre todo de tiempos de acceso y respuesta a la base de datos).

Las dos maneras más comunes de manejar esto es mediante un almacenamiento basado en tiempo o un almacenamiento basado en cambio.

El primero almacenará los valores en la base de datos cada x tiempo, que será definido. Puede ser que el valor haya cambiado o no, en cualquier caso el valor será almacenado.

El segundo es que se almacenará el valor cada vez que este cambie en un intervalo definido. De esta manera siempre se tendrán valores diferentes.

El segundo método resulta en general más eficiente en cuanto a consumo de espacio y es el que se emplea para almacenar los datos en este caso.

2.1.6 La base de datos LHC Logging

La aplicación no accederá directamente a la base de datos del sistema SCADA por cuestiones de seguridad y de rendimiento. Por ello el SCADA se ha configurado de manera que cada noche publique la lista de valores de los datapoints del archivo que empleará la aplicación en una base de datos diferente. Esta base de datos es *LHC Logging* y allí se almacena información a largo plazo y de forma masiva.

La aplicación accederá a LHC Logging para recuperar los valores de los datapoints diariamente y los importará dentro de la base de datos de la aplicación para realizar los posteriores cálculos.

2.1.7 Conclusiones

En este apartado se ha pretendido acercar al lector a los diferentes entornos con los que trabajará este proyecto. Se ha tratado de simplificar y sintetizar al máximo de manera que sea lo más liviano posible.

Es complicado explicar en estas pocas líneas la complejidad de los diferentes entornos y además está fuera del ámbito de este documento, pero una idea general es necesaria para poder profundizar en los próximos capítulos.

2.2 Requisitos funcionales

La aplicación ha de responder a numerosas necesidades que se le han encomendado, entre ellas:

1. Proporcionar una interfaz web que muestre la evolución de los consumos a lo largo de los días de manera gráfica.
2. La interfaz también ha de mostrar los consumos mensuales.
3. Debe permitirse la exportación de los resultados (por ejemplo CSV).
4. Debe poder seleccionarse el Consumer, el mes y el año para mostrar los datos.
5. Cálculo y visualización de la potencia calculada a través del consumo

6. Debe contar con una parte administrativa en la que sea posible:
    a. Añadir nuevos Consumers
    b. Añadir nuevos Feeders
    c. Añadir nuevos Datapoints
    d. Borrar Consumers
    e. Borrar Feeders
    f. Borrar Datapoints
    g. Modificar el árbol (p. ej. Cambiar el padre de un Consumer)
    h. Exportación masiva de datos
    i. Relanzar el cálculo de consumos.
    j. Exportar o visualizar el árbol
    k. Añadir correcciones puntuales a los datos
7. Esta aplicación administrativa deberá estar debidamente restringida.

2.3 Requisitos no funcionales

A la hora de realizar la aplicación se han encontrado numerosas limitaciones que previamente no se habían previsto. Un ejemplo de estas es el hecho de que por un tema de rendimiento no se acceda directamente a las base de datos del archivo y se haya realizado a través de la base de datos LHC Logging. Es fundamental que para el funcionamiento de WebEnergy no se comprometa el correcto funcionamiento del SCADA y que haga peligrar la supervisión.

También la elección de las tecnologías empleadas responde a requisitos internos de la organización: por un lado, éstas tienen que estar soportadas dentro del CERN, como es el caso de Java o Spring. WebEnergy debe realizarse de manera que en el futuro otras personas puedan acceder a su código para realizar modificaciones o actualizaciones y por ello se ha escogido un lenguaje ampliamente utilizado en la organización.

Por otro lado, las librerías a emplear deberían ser de uso libre y gratuito y lo más estándar posible. Java Server Faces es un estándar de facto en el desarrollo JEE y Primefaces es una de las librerías de componentes más importantes (y potentes para JSF). Primefaces soporta también gráficos a través de jqPlot, lo que facilitará el desarrollo. También Hibernate es el ORM más utilizado en Java junto con JPA.

Se ha incidido bastante en esta documentación en hacer el código mantenible y lo más inmune a cambios posible. No solo porque el gestor de base de datos pueda cambiar en el futuro, sino porque el sistema SCADA no será eterno y en unos años tendrá que ser también actualizado y/o reemplazado por uno más

moderno. Esto ha ocurrido recientemente y se sustituyó el antiguo sistema SCATEX de EFACEC por WinCC OA de Siemens. Obviamente cuando esto vuelva a suceder, es importante que el impacto en la aplicación sea el menor posible, aunque nunca será cero.

La arquitectura de la aplicación garantizará que la capa que soporte la mayor parte de los cambios sea la de persistencia (las bases de datos de los diferentes sistemas SCADA variarán), por lo que se ha tratado de desacoplar las diferentes capas de la aplicación lo más posible.

3 Solución desarrollada

3.1 Fundamentos

3.1.1 Introducción

El corazón de WebEnergy es la manera en que se estructura la información y se calculan los resultados, a fin de cuentas éstos son lo más importante. El objeto de esta sección será explicar cómo se mueve la información entre los diferentes entornos, cómo se estructura en WebEnergy y finalmente cómo se calculan los resultados desde un punto de vista teórico.

3.1.2 Una visión global

La información de los consumos de energía contenidas en los valores de los datapoints ha de ser importada dentro de la base de datos de la aplicación. Como se comentaba en el capítulo anterior, la información se genera en el sistema SCADA que va insertando la misma en su base de datos de archivo.

Dado que WebEnergy no puede ni debe acceder a esta base de datos directamente, el SCADA publicará estos valores en la base de datos LHC Logging. WebEnergy no podrá directamente extraer los datos de LHC Logging, creando una conexión y consultas SQL directamente, sino que se realiza a través de unas librerías especiales que se insertan en los proyectos Java (para este caso).

El sistema SCADA publica estos datos al filo de la media noche una vez al día por lo que la recuperación de los mismos se ha fijado en una tarea programada en torno a las 12 y media de la madrugada.

Los datos en bruto serán insertados en la base de datos de la aplicación y una vez que esto se ha realizado, se puede lanzar el cálculo de los consumos. Se puede ver un esquema del flujo de datos entre las distintas bases de datos en la Figura 3.

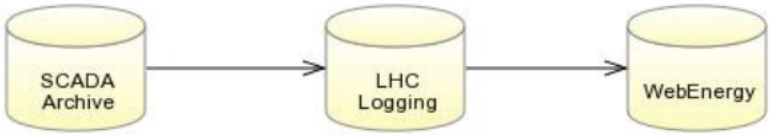

Figura 3: Flujo de datos

Para realizar este cálculo se necesita la estructura de la red eléctrica y cómo combinar estos valores, que se abordará a continuación.

3.1.3 Estructurando los datos

De por sí, los datos ofrecen información totalmente desestructurada que no arroja luz sobre los posibles consumos de energía. Esta información debe ser combinada de manera que se consigan calcular consumos para los diferentes lugares de interés.

Existirán en WebEnergy cuatro tipos de entidades:

3.1.3.1 Datapoints

Los datapoints se emplean, como se había comentado anteriormente, para almacenar valores o medidas de diversos parámetros en los equipamientos que se controlan. Estos valores sirven para monitorizar estos equipos y supervisar su correcto funcionamiento.

Existen tres grupos de datapoints:

- **Digitales**: que almacenan un valor digital (un estado: 0 ó 1, encendido o apagado)
- **Analógicos**: que almacenan un valor analógico como puede ser la temperatura, voltaje, corriente.
- **Contadores**: que almacenan un valor normalmente creciente y acumulativo (como el consumo energético)

Los datapoints que empleará WebEnergy para el cálculo de los consumos en los diferentes puntos de interés, serán casi en su totalidad de tipo contadores. Particularmente aquellos datapoints definidos en el SCADA con sufijo Ea+ y Ea-.

3.1.3.2 Feeders

Estos datapoints siempre van por pares, es decir, por cada Ea+ existirá un Ea-. Los datapoints positivos representarán la *energía entrante* en el dispositivo y los negativos, la *energía saliente*.

Por tanto para saber exactamente la energía consumida por el dispositivo en particular o conjunto de dispositivos medidos por el par de datapoints, tan solo será necesario hacer una resta:

Energía entrante – Energía saliente = Consumo de energía

O lo que es lo mismo, Ea+ menos Ea-.

Dado que estos datapoints siempre vienen por pares, en WebEnergy se ha creado una nueva entidad para representar cada par. A estas nuevas entidades se las llamará *Feeders*.

Por ejemplo, la resta de los datapoints *EMD205/2E_Ea+* y *EMD205/2E_Ea-* será el valor que tomará el feeder *EMD205/2E*.

3.1.3.3 Consumers

Los feeders no son entidades interesantes para las que desglosar ni calcular consumos. Son solo un cálculo intermedio.

Los feeders se combinarán entre sí (sumándose y restándose) para crear lo que se llamarán *Consumers*. La manera en que se combinarán los feeders responde a la estructura de la red eléctrica del CERN y sus complejidades y queda fuera del objeto de este texto.

Los Consumers ya no serán entidades abstractas sino que se corresponden a elementos reales o tangibles (un experimento, un edificio, un complejo, determinadas maquinarias etc.) y que cuentan con un interés suficiente como para calcular sus consumos. Estos valores podrán ser susceptibles de ser mostrados en la interfaz de la aplicación y siempre se almacenarán en la base de datos de la aplicación.

Como ejemplo, en el momento de escribir este texto, el consumer *Computer Centre*, se compone de los siguientes Feeders:

EFD101*43

EMD102*43

EFD103*43

EMD201*43

EMD101*43

El nombre de los Feeders no es aleatorio sino que responde a una convención establecida para dar información sobre los mismos. Por ejemplo, la (E) significa que es un equipamiento eléctrico; la (M) que es un motor, la (D) viene de *départ* (salida) y el sufijo (*43) indica el lugar donde se sitúa (el asterisco significa que es en el sitio de Meyrin).

3.1.3.4 Super Consumers

Los Consumers se podrán combinar entre ellos de manera similar a la que hacen los Feeders para crear lo que se llaman *Super Consumers*. Estos no son más que Consumers compuestos por la suma de otros Consumers más pequeños y que estarían contenidos en ellos. Los Super Consumers también pueden asociarse a elementos reales y normalmente el cálculo de sus consumos tendrá un interés superior. Dado que un Super Consumer es en esencia un Consumer, podrá haber Super Consumers de Super Consumers de manera que el árbol creado pueda crecer lo necesario. En la Figura 4 se puede observar una representación de estas estructuras.

Como ejemplo, el Super Consumer *CERN* (que es el nodo raíz y el consumidor resultante de todo el cálculo) se compone de los Super Consumers *LHC*, *SPS* y *PS-Meyrin*.

A su vez, el Super Consumer *LHC* se compone de los Super Consumers, *General Services*, *Experiments* y *Machine*.

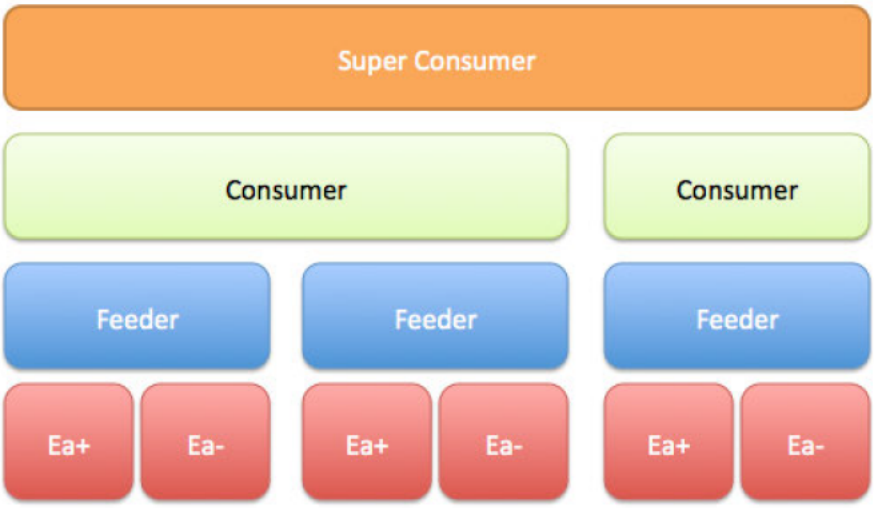

Figura 4: Representación de las diferentes entidades

Por tanto se puede llegar a la conclusión que toda la estructura de la red eléctrica se puede representar con una estructura de árbol. Sumando y combinando los distintos Consumers, se podrá llegar a calcular el consumo total del nodo raíz y por lo tanto, de todo el CERN.

En la imagen anterior se puede observar una pequeña representación gráfica de cómo los datapoints se combinan en Feeders, los Feeders en Consumers y los Consumers se agrupan en Super Consumers, así hasta llegar a lo más alto del árbol.

3.1.4 El modelo de datos

3.1.4.1 Representación

Para representar la estructura de árbol, se empleará la base de datos de la aplicación. Ésta contará con varias tablas para representar los datapoints, los feeders y los consumers en la misma. El aspecto del diagrama entidad-relación para estas tablas se tiene en la Figura 5:

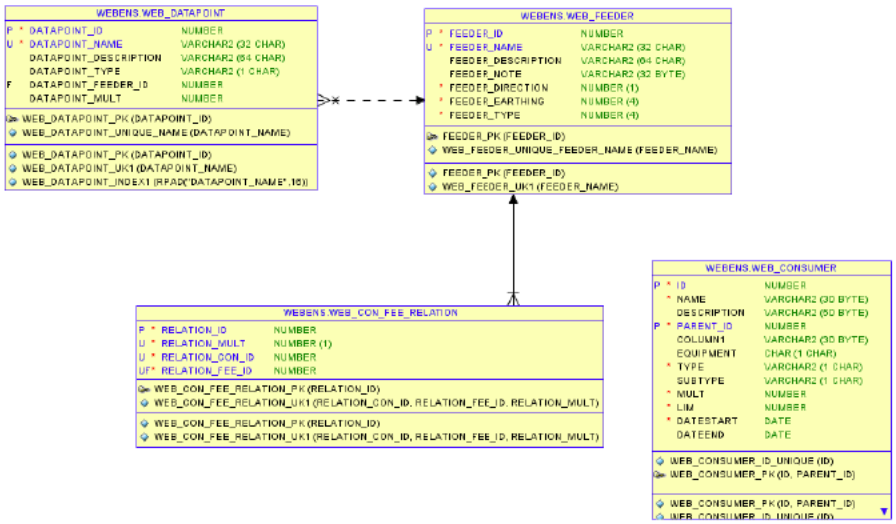

Figura 5: Modelo de datos

Gracias a estas tablas se podrá establecer las relaciones entre las diferentes entidades.

3.1.4.2 Tabla WEB_DATAPOINT

La tabla WEB_DATAPOINT en la Figura 6 almacenará los nombres de todos los datapoints que se emplearán para realizar el cálculo. Esta tabla estará relacionada directamente con la tabla WEB_FEEDER por lo que se definen en cada entrada el id del Feeder al que pertenecen los datapoints.

También se almacenará su id, su nombre y un multiplicador que servirá para indicar en el cálculo si es un datapoint positivo o negativo, de manera que su energía será sumada o restada.

DATAPOINT_ID	DATAPOINT_NAME	DATAPOINT_DESCRIPTION	DATAPOINT_TYPE	DATAPOINT_FEEDER_ID	DATAPOINT_MULT
3667	EFD101*43_Ea+	EFD101*43_Ea+	+	29	1
4191	EFD101*43_Ea-	EFD101*43_Ea-	-	29	-1
1340	EFD101*59_Ea+	EFD101*59_Ea+	+	441	1
1341	EFD101*59_Ea-	EFD101*59_Ea-	-	441	-1
3419	EFD101*6_Ea+	EFD101*6_Ea+	+	57	1

Figura 6: Aspecto de la tabla WEB_DATAPOINT

3.1.4.3 Tabla WEB_FEEDER

La tabla WEB_FEEDER en la Figura 7 almacenará los Feeders que se han definido en WebEnergy. Se identificarán por su feeder_id (campo clave de la tabla) aunque también el campo feeder_name será único.

En esta tabla existen otros tres campos (feeder_direction, feeder_earthing y feeder_type) que solo tomarán valores de 1 y -1. Estos valores servirán para saber el signo final del feeder. Los tres valores se multiplicarán de manera que resultará positivo o negativo (1 ó -1). El significado de estos valores está relacionado con cuestiones eléctricas y de la estructura de la red.

FEEDER_ID	FEEDER_NAME	FEEDER_DESCRIPTION	FEEDER_NOTE	FEEDER_DIRECTION	FEEDER_EARTHING	FEEDER_TYPE
7	END403/4E	END403/4E	(null)	1	1	1
8	END601/6E	END601/6E	(null)	1	1	1
9	END604/6E	END604/6E	(null)	1	1	1
10	END303/7E	END303/7E	(null)	1	1	1
11	END606/2E	END606/2E	(null)	1	1	1

Figura 7: Aspecto de la tabla WEB_FEEDER

3.1.4.4 Tabla WEB_CONSUMER

La tabla WEB_CONSUMER en la Figura 8 almacena los Consumers y Super Consumers definidos en la aplicación. Estos Consumers se almacenan en forma de árbol, empleando una estrategia de id y parent_id para este cometido.

ID	NAME	DESCRIPTION	PARENT_ID	TYPE	MULT
581	LHC	LHC 3.5 TeV	721	C	1
582	SPS	SPS Prevessin	721	C	1
583	PS - Meyrin	PS Complex - Meyrin	721	C	1

Figura 8: Aspecto de la tabla WEB_CONSUMER

De esta manera cada Consumer tendrá un id asociado y también tendrá un Consumer padre dado por su parent_id, representando una estructura de árbol en la base de datos.

Los Consumers y los Feeders se relacionan por medio de una tabla intermedia WEB_CON_FEE_RELATION. Esta tabla servirá para representar una relación N a N (muchos a muchos) en la que un Consumer se compondrá de muchos feeders, pero donde también un Feeder puede estar en muchos Consumers.

En esta tabla también habrá un campo llamado RELATION_MULT (multiplicador de la relación). Este dato representará si el Feeder sumará o restará respecto a la energía consumida por el Consumer. Dependiendo de la estructura de la red, algunos feeders puedes sumar en determinados Consumers y restar en otros.

3.1.5 El cálculo de los consumos

3.1.5.1 Importación y cálculo de los valores diarios de los datapoints

Cada noche, después de que el SCADA ha exportado los valores de los datapoints a la base de datos LHC Logging, la aplicación accederá a esta base de datos para importar los valores de todos los datapoints (definidos en la tabla WEB_DATAPOINT) para el último día.

Debido a que el sistema de archivo en el SCADA se ha fijado como *event-based* (basado en eventos), solo se almacenará el valor del datapoint en el archivo si éste ha cambiado en un determinado intervalo. Por lo tanto si no ha habido consumo de energía (el valor del datapoint no ha variado) no habrá ningún valor que importar para el datapoint en ese día.

Para realizar el cálculo, se toma la lista de valores del día para el datapoint dado y se añade el último valor existente en la base de datos para el mismo. Por ejemplo, para el datapoint EKD204/5E_Ea+ para calcular su valor para el día 21/08/2015 se tomarán los valores del día 21 y el último del día 20, como se ve en la Figura 9.

	NAME	DATETIME	VALUE	ID
1	EKD204/5E_EA+	21/08/2015 23:59:59	4368360	482407
2	EKD204/5E_EA+	21/08/2015 00:19:59	4360476.5	482406
3	EKD204/5E_EA+	20/08/2015 23:59:59	4360359	481737
4	EKD204/5E_EA+	20/08/2015 00:00:01	4352474.5	481736
5	EKD204/5E_EA+	19/08/2015 23:40:00	4352356.5	481271

Figura 9: Valores para EKD204/5E_Ea+

Una vez que se tiene la lista preparada, al valor más reciente (temporalmente hablando) se le restará el valor más antiguo. En el ejemplo, el valor del datapoint para el día 21/08/2015 sería el siguiente:

4368360 – 4360359 = 8001 kWh

Para el caso del datapoint Ea-, en la Figura 10 se observa que no hay valores para el día 21 de agosto, por lo tanto significa que su valor no ha cambiado y será cero.

	NAME	DATETIME	VALUE	ID
1	EKD204/5E_EA-	20/08/2015 11:40:00	0.046999998390674591	481738
2	EKD204/5E_EA-	06/08/2015 10:02:46	0.046999998390674591	475136
3	EKD204/5E_EA-	16/07/2015 12:09:24	0.046999998390674591	465615
4	EKD204/5E EA-	02/07/2015 11:05:10	0.046999998390674591	458455

Figura 10: Valores para EKD204/5E_Ea-

3.1.5.2 Posibles errores

Los valores para los datapoints dado que son contadores, siempre irán *in crescendo*. Esto siempre será así excepto en dos casos principales:

1. Reseteo del contador
2. Cambios en el equipo

El primer caso se da cuando el valor del contador en el equipo llega a su máximo y retorna a cero. Como ejemplo vale la analogía se puede hacer con un antiguo cuentakilómetros de coche. Después de alcanzar el valor de 999.999 km, el cuentakilómetros retornará a 0. Obviamente esto será un valor engañoso a la hora de calcular los consumos, aunque es fácil de detectar ya que en el cálculo el valor más reciente será inferior al valor más antiguo y eso es imposible.

En la siguiente gráfica se tiene un ejemplo del aspecto de la evolución del valor de un datapoint cuando se produce un reseteo del contador. Se puede observar cómo hasta las 00:25 el valor del datapoint va creciendo, para descender hasta cero de manera brusca. Inmediatamente después vuelve a ascender.

Si se calculase el valor para el mismo, se tendría un valor negativo ya que el último valor del intervalo es inferior al primero.

Para solventar este problema el cálculo del valor se divide en dos: antes del cero y después del cero. Se calculará la diferencia entre los valores antes y después y se sumarán para tener el consumo real y evitar el problema del reseteo. En rojo en la Figura 11 se pueden observar los valores reales del crecimiento del contador para el datapoint.

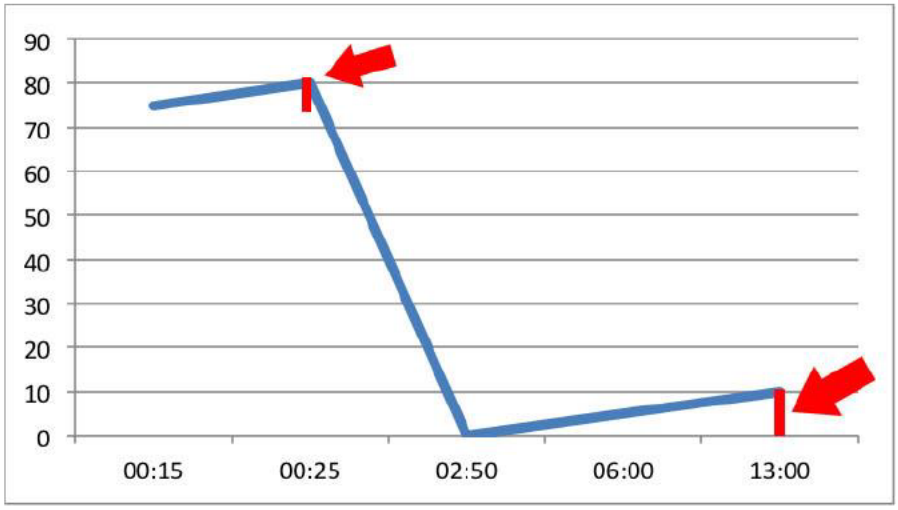

Figura 11: Ejemplo de reseteo de contador

El segundo caso (cambios en el equipo) es más difícil de detectar y de resolver. Esto puede darse cuando un equipo se reemplaza por otro, se resetea su memoria o se realizan diferentes cambios en el mismo que conlleven un cambio en los valores de los contadores. Por ejemplo si se sustituye el aparato por otro con diferentes valores de los contadores, se podrán provocar picos (si el valor del contador es súbitamente muy superior) o valores negativos como de manera similar a la anterior.

Como se comentaba, eso es difícil de resolver de manera automatizada y serán los usuarios administradores de la aplicación los que han de corregir estas inevitables desviaciones aplicando correcciones a los valores. La aplicación sí proporcionará herramientas para la detección de este tipo de picos estableciendo una serie de valores límite (por ejemplo del 1000%) en la variación de los consumos de los Consumers. Cuando se produce este disparo del consumo se puede atribuir a algún tipo de problema derivado de algún tipo de intervención sobre los equipos y habrá que comprobar que esto efectivamente ha sido así.

3.1.5.3 Cálculo de los valores de los Feeders

El cálculo del valor de los feeders se realiza en una etapa anterior al cálculo de energía consumida por el árbol de los consumers. Para estos valores se guardan en la tabla PSEN_FEEDER_CONSUMPTION como paso intermedio.

Cada Feeder tendrá un valor diario y para calcularlo se han de tener los valores de los datapoints Ea+ y Ea- calculados previamente. Luego se procederá a su resta.

Si se toma de ejemplo los datapoints anteriores para calcular el feeder EKD205/5E, el valor para el feeder será el siguiente:

8001 kWh – 0 kWh = 8001 kWh

Si se observa la tabla PSEN_FEEDER_CONSUMPTION en la Figura 12, para este Feeder se puede ver cómo el valor coincide:

	ID	NAME	IDDAY	VALUE
1	147204	EKD204/5E	5712	8001
2	146944	EKD204/5E	5711	8002.5
3	146684	EKD204/5E	5710	7641
4	146424	EKD204/5E	5709	7666
5	146164	EKD204/5E	5708	7747

Figura 12: Valores para el feeder EKD204/5E

Una de las razones por las que no se incorporan los Feeders al árbol de Consumers es porque éstos no se emplean de manera única en los Consumers, sino que pueden emplearse repetidamente en diferentes por lo que también resulta más manejable el guardar sus valores en la base de datos aunque solo sean empleados como paso intermedio para hallar los consumos. Recordemos que la relación en el modelo entidad-relación de la Figura 5 era *N* a *N*.

3.1.5.4 Cálculo del árbol de Consumers.

Para realizar el cálculo de los consumos se necesitará recorrer el árbol de Consumers representado en la base de datos. El algoritmo para el cálculo cargará, en una estructura de datos en memoria, el árbol de Consumers, directamente desde la tabla WEB_CONSUMER.

El grafo se recorrerá en *post-orden*, para lo que se empleará una *pila*[2]. Esta estructura de datos permitirá realizar el recorrido por niveles de abajo arriba, de manera que para calcular el nivel superior ya se tengan los valores de todos los elementos del nivel inmediatamente inferior.

[2] Una *pila* es una estructura de datos de tipo LIFO (last-in, first-out) de la que se recuperarán en primer lugar los últimos valores introducidos. Sería exactamente lo opuesto a una *cola*, de tipo FIFO (first-in, first-out).

Esto se consigue recorriendo el árbol de la Figura 13 de arriba abajo por niveles y metiendo los nodos en la pila. Por tanto se comenzará por el nodo raíz, le seguirá el nivel 2 y finalmente el nivel 3. La pila tendrá el siguiente aspecto:

$$
\begin{array}{c}
H \\
G \\
F \\
E \\
D \\
C \\
B \\
A \\
\hline
\end{array}
$$

Empleando el método *pop()* de la pila, se comenzarán a recuperar los *nodos hoja* (se corresponderán con Consumers sencillos, es decir, aquellos compuestos únicamente por Feeders). De esta manera se comenzarán a calcular los consumos para los nodos H, G, F y E.

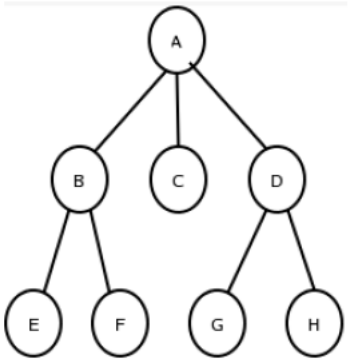

Figura 13: Recorrido del árbol

Para cada uno de estos nodos hoja, se accederá a la lista de Feeders de los que se compone y por cada uno de estos, a los valores de los Feeders previamente calculados en la tabla PSEN_FEEDER_CONSUMPTION. Con todos estos valores se podrá realizar el cálculo de su consumo, que se almacenará en un HashMap.

A la hora de calcular el consumo total del Consumer, se ha de tener en cuenta el valor de los feeders que lo componen y el resultado de sus campos

(feeder_direction, feeder_earthing y feeder_type) que solo tomarán valores de 1 y -1.

También se ha de tener en cuenta el valor del campo relation_mult en la tabla WEB_CON_FEE_RELATION. Los tres campos anteriores y el campo realtion_mult se multiplicarán entre sí para determinar el sentido (positivo o negativo) del Feeder en el Consumer dato, de la siguiente manera (Listado 3):

```
foreach (Feeder perteneciente a Consumer) {
ConsumoDelConsumer += valorFeeder * multiplicadorRelacion *
direccionFeeder * earthingFeeder * tipoFeeder
}
```

Listado 1: Cálculo del consumo para un Consumer

Siendo ConsumoDelConsumer la variable que acumula el consumo del Consumer a través de sus diferentes feeders.

Hasta ahora se ha visto cómo se calcula el valor para los Consumers, no para los Super Consumers. Anteriormente se ha comentado que el cálculo completado para los consumers se almacena en un HashMap, con clave su id y valor su consumo.

Gracias a la manera en que se recorre el árbol emplenado la pila que se ha comentado antes, los valores de los Consumers hijos estarán ya calculados e insertados en el HashMap, por lo que podrán ser recuperados de la siguiente manera (Listado 2):

```
for (Consumer perteneciente a SuperConsumer) {
ConsumoSuperConsumer += ConsumerHash.getConsumer().getConsumo *
ConsumerHash.getConsumer().multiplicador
}
```

Listado 2: Cálculo del consumo de un Super Consumer

El Super Consumer no será más que la suma de Consumers (o Super Consumers, a medida que se avance en el árbol). Los Consumers también tienen un multiplicador (normalmente siempre será +1) pero de igual manera que con los feeders, un Consumer puede ser negativo para su Super Consumer, aunque a diferencia de los feeders, no puede estar repetido en el árbol y solo tendrá un Super Consumer padre.

Una vez que el consumo para el Super Consumer se ha calculado, se insertará en el HashMap para que colabore con el cálculo de su Consumer padre, de esta manera se irá ascendiendo en el árbol hasta llegar al Super Consumer CERN, que es el nodo raíz y que representará el consumo energético producido en todo el CERN.

Por supuesto, a medida que se realizan estos cálculos de consumos diarios, se van insertando en la base de datos de la aplicación que se empleará para mostrar su evolución. La tabla donde se insertan los consumos es la tabla CEM_E_P_DAILY, que tiene el aspecto de la Figura 14:

ID_MONTH	ID_CONSUMER	ID_DAY	ENERGY_DAILY
188	2162	5713	172296
188	2161	5713	304416
188	669	5713	781856
188	605	5713	34060
188	604	5713	21544
188	597	5713	60349.18945...

Figura 14: Aspecto de la tabla CEM_E_P_DAILY

3.1.6 Interacción entre las distintas aplicaciones

WebEnergy no se compone por una sola aplicación, sino que está dividida en tres aplicaciones diferentes e independientes entre sí. Estas aplicaciones se desplegarán de manera individual en el contenedor de servlets (se está empleando Apache Tomcat 8 para el sistema de producción).

WebEnergy se puede dividir en:

1. WebEnergy
2. WebEnergyAdmin
3. WebEnergyImport

WebEnergy es la aplicación pública, donde se generarán los reportes con los diferentes consumos y las gráficas con las evoluciones.

WebEnergyAdmin proporciona una serie de herramientas administrativas para gestionar la aplicación (creación de consumers, feeders y datapoints, gestión de correcciones, etc.) pero también se encargará de realizar el cálculo de los consumos y de exponer una pequeña API REST bajo JSON.

WebEnergyImport es de las tres, la aplicación más pequeña. Únicamente se encarga de acceder a la base de datos LHC Logging para recuperar los valores de los datapoints en el día y los inserta en la tabla PSEN_DATA_RAW de la base de datos de WebEnergy. Esto se realiza todas las noches mediante una *Scheduled Task* de Spring. Una vez insertados, llamará a la API de WebEnergyAdmin mediante un request http para lanzar el cálculo diario.

La importación de los datos se ha realizado en otra aplicación porque el acceso a la base de datos LHC Logging se hace a través de una librería especial que podría generar incompatibilidades con las utilizadas en WebEnergyAdmin en el caso de desplegarse en el mismo archivo war.

Las tres aplicaciones accederán de una manera u otra a la base de datos principal de la aplicación, pero es WebEnergyAdmin la que realiza las tareas de gestión y de cálculo más complejo, además de exponer una capa de servicios de integración (API REST).

3.2 Arquitectura de la aplicación

3.2.1 Introducción

En esta sección se va a realizar un acercamiento a la arquitectura de la aplicación sobre la que se va a trabajar. Como se comentaba anteriormente, en la aplicación se emplean varios frameworks para su desarrollo. Se tratará de explicar también cómo influyen estos frameworks y dónde actúan en la arquitectura de la aplicación.

3.2.2 Principios arquitectónicos de la aplicación

La aplicación se ha dividido en tres capas fundamentales como se puede ver en la Figura 15: la capa de acceso a datos, la capa de negocio y la capa de presentación. La ventaja principal de este modelo es que el desarrollo se puede

llevar a cabo en varios niveles y en el caso de tener que realizar cambios en alguno de los niveles, no se impactan los otros ya que están desacoplados.

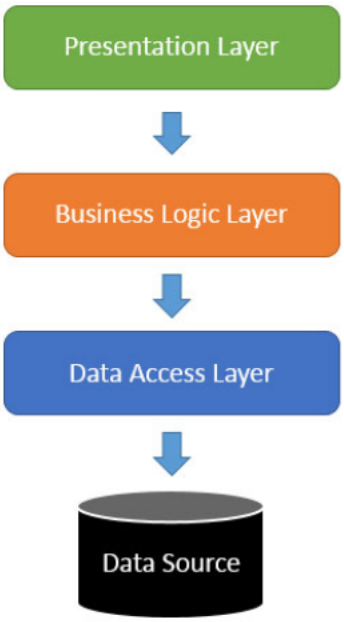

Figura 15: Modelo por capas

Otra de las ventajas es que se puede distribuir el trabajo por niveles, estando cada grupo de trabajo totalmente abstraído del resto de niveles, de forma que basta con conocer la API existente entre los mismos.

3.2.2.1 La capa de datos

Es la capa que está en contacto directo con la base de datos y se encarga de acceder a la misma. Estará formada por uno o más gestores de bases de datos que realizan tanto la extracción como la persistencia de datos en la base de datos.

Para acceder a los datos se empleará el framework Hibernate. Hibernate es una herramienta de mapeo objeto-relacional para Java que facilita el mapeo entre los atributos de una base de datos relacional y las entidades (objetos POJOs) dentro de Java. Por cada tabla de la base de datos se creará una entidad que no es más que un Bean de Java que encapsulará los datos.

Además Hibernate facilita la realización de las consultas a la base de datos ofreciendo un lenguaje propio de consulta (HQL, Hibernate Query Language), gestionando las conexiones mediante una pool de conexiones.

3.2.2.2 La capa de negocio y la arquitectura orientada a servicios (SOA)

La capa de negocio almacena la lógica de negocio de la aplicación. Aquí será donde se realicen la mayor parte del procesamiento y es la capa de mayor complejidad.

La capa de negocio se comunica con la capa de presentación para recibir las solicitudes de los usuarios y con la capa de datos para solicitar a la misma el acceso a la base de datos. La capa de negocio estará formada en este caso por **servicios**. Un servicio encapsulará una funcionalidad que podrá ser reutilizada en diferentes lugares del código. Precisamente la idea detrás de la creación de los servicios es la **reutilización sistemática de código**. Cada servicio solo ha de ser testeado una vez y utilizado sistemáticamente. Esta es la gran ventaja de una **arquitectura orientada a servicios (SOA)** [1] con respecto a las antiguas arquitecturas monolíticas.

Además otro de los conceptos detrás de la creación se los servicios, es la de la **modularidad**. La modularidad es la propiedad que permite dividir una aplicación en partes más pequeñas *divide y vencerás* siendo cada una de las partes (llamadas módulos) tan independiente de las otras como sea posible.

Esto no es más que desacoplar más aun la aplicación. Al crear servicios separados ya no solo se desacoplarán las capas de datos, negocio e interfaces, sino que los propios servicios estarán desacoplados entre sí.

Esto conlleva un beneficio inmediato a parte de la reutilización y que es que la aplicación no solo escalará adecuadamente de manera vertical (con la separación en capas) sino que además escalará bien de manera horizontal (con la creación de más servicios).

Es importante señalar que **un servicio puede emplear otros servicios**, siendo posible la reutilización de los mismos dentro de la propia capa de servicios, no siendo necesario que estas reutilizaciones sean entre diferentes niveles, sino que gracias a la modularidad de la capa de servicios, unos componentes pueden llamar a otros.

No obstante dentro de una arquitectura orientada a servicios esto debe ser puntualizado ya que la capa de negocio (también llamada capa de servicios) puede descomponerse a su vez en varias capas de servicios:

3.2.2.2.1 Capa de servicios de acceso a datos

La capa de servicios de acceso a datos es la capa de servicios más cercana a la capa de datos donde se situarán los DAOs de la aplicación. Básicamente se encargará de comunicarse con los DAOs para obtener y persistir datos, siendo esta su única misión. De esta manera otros servicios que requieran el acceso a datos, no lo harán a través de los DAOs, sino que lo harán a través de esta capa de servicios.

En la Figura 16 puede observarse la descomposición de la capa de servicios dentro de una aplicación orientada a servicios. Se puede observar cómo la capa de acceso a datos (en rojo) es la única que accede a la base de datos. Hay que puntualizar que lo hará a través de los DAOs en la capa de datos.

Por otro lado, en azul, se tiene la capa de servicios de negocio que se comunica con la capa de servicios de acceso a datos. De esta manera, no necesitará bajar a la capa de datos directamente (donde se albergan los DAOs).

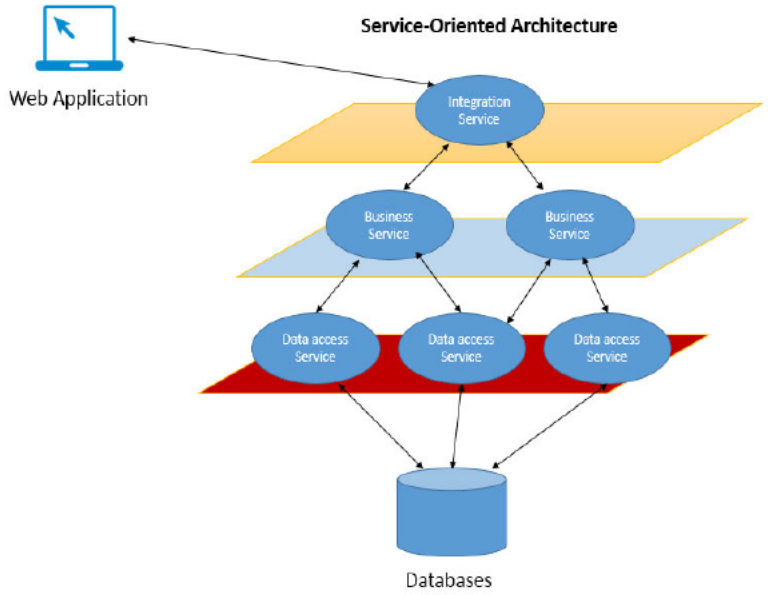

Figura 16: Aplicación orientada a servicios

3.2.2.2.2 Capa de servicios de negocio

La capa de servicios de negocio la podemos ver en azul en la imagen. Estos servicios normalmente necesitarán acceder a la base de datos (normalmente a varias tablas diferentes) y lo harán a través de los servicios de la capa de acceso de datos. Por tanto hay una diferencia fundamental entre las dos capas. Mientras los servicios de la capa de acceso a datos estarán relacionados cada uno con un DAO y por tanto con una tabla (aunque puede haber alguna excepción), los servicios de negocio pueden necesitar acceder a varias tablas de la base de datos y lo harán empleando varios servicios de la capa de acceso a datos.

Los servicios de la capa de negocio se encargan de abstraer funcionalidades de negocio. Estas funcionalidades tendrán un sentido claramente funcional en el negocio, pudiendo identificarlas con actividades concretas (un servicio de negocio normalmente involucra varios servicios y de alguna manera actúa como patrón *façade*).

3.2.2.2.3 Capa de servicios de integración

Esta es la última capa que se distinguirá dentro de los servicios y corresponde al nivel más alto de la misma y por tanto más alejado de la capa de datos. La capa de servicios de integración podrá entre otras cosas, comunicarse con las interfaces o la de la publicación de una API RESTful, por ejemplo.

La capa de servicios de integración proporciona eso mismo, integración con otros sistemas. Mediante una API RESTful, se podrán establecer comunicaciones con otras aplicaciones que incluso estén programadas en otras plataformas. También se podrán establecer interfaces con librerías Javascript como knockoutJS o AngularJS que se comuniquen directamente con la API REST para dar funcionalidad a la interfaz, empleando esta capa de integración como backend.

En definitiva, las aplicaciones de esta capa pueden ser multitud. Es importante destacar que la capa de integración se comunica directamente con la capa de servicios de negocio y expondrá al exterior los servicios de negocio que se deseen, mediante la publicación de webservices ya sean SOAP o REST.

En el caso de WebEnergy, existe una capa de servicios de integración para la comunicación con otras aplicaciones. Es una pequeña API RESTful, pero que no pretende sustituir la capa de presentación.

3.2.2.2.4 La capa de negocio y Spring

La capa de negocio hará un uso extensivo de Spring Framework. Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión del control para Java mediante inyección de dependencias (DI).

Spring será la piedra angular que permitirá inyectar dependencias en las diferentes clases, en las diferentes capas. Por ejemplo, los DAOs podrán ser inyectados en la capa de servicios de acceso a datos y a su vez estos servicios podrán ser inyectados en otros servicios o en la capa de presentación (ManagedBeans de JSF).

La gran contribución y ventaja de usar la inversión del control es que las clases no instanciarán sus objetos, sino que el contenedor de Spring se encargará de ello y de inyectar las dependencias necesarias en las clases, liberando al programador de la responsabilidad de emplear el operador *new* en las clases, lo que acoplaría fuertemente las clases.

3.2.2.3 La capa de presentación

La capa de presentación se encargará de la comunicación con los usuarios de la aplicación y de mostrar la interfaz de la misma. La capa de presentación se comunicará directamente con la capa de servicios (o de negocio) para realizar las operaciones necesarias. Es muy importante destacar que, a pesar de que las funciones del negocio residen en otro nivel, para el usuario es transparente la forma de operar.

Como framework web se empleará JSF (Java Server Faces) que simplificará el desarrollo de interfaces de usuario en aplicaciones Java. Java Server Faces contiene un conjunto de APIs para representar componentes de interfaz de usuario así como *beans* administrados.

Entre las ventajas de emplear JSF se encuentra la posibilidad de emplear librerías de componentes más avanzados como MyFaces, ICEFaces o PrimeFaces. Estas librerías contienen componentes muy elaborados que permiten simplificar aún más el desarrollo web.

Java Server Faces también trabajará conjuntamente con Spring, permitiendo la inyección de dependencias dentro de los managedbeans de JSF. De esta manera los servicios que necesiten ser accedidos desde las interfaces serán inyectados en las clases por Spring.

3.2.3 Seguridad

La seguridad debe ser una parte importante a la hora de desarrollar una aplicación. Para proteger los diferentes accesos a las aplicaciones se ha empleado Spring Security.

Spring Security permitirá configurar el acceso contra el directorio activo *cern.ch*, de manera que solo las cuentas NICE[3] podrán acceder a WebEnergy. Para acceder a la parte administrativa además de contar con una cuenta NICE, se deberá pertenecer a un grupo de usuarios administradores definidos mediante la herramienta egroups.cern.ch

Por otro, lado Spring Security se ha configurado de manera que fuerce las conexiones por el canal seguro HTTPS, para lo que se ha instalado un certificado en el servidor, proporcionado por la *CERN Certification Authority*.

3.3 Diseño de la aplicación para facilitar la mantenibilidad

3.3.1 Introducción

A lo largo de las siguientes páginas se estudiarán las decisiones de diseño empleadas en las diferentes capas de la arquitectura, separándose por capa de datos, capa de negocio y capa de presentación.

Se comentarán también los principios de desarrollo en las que se apoyan las decisiones de diseño de software que se han aplicado.

Uno de estos principios que es vital para la extensibilidad de la aplicación, y es una de las decisiones importantes que se han adoptado en el diseño de la aplicación, es el de la *programación contra interfaces* y no contra las implementaciones. Se verán a lo largo de las próximas páginas los beneficios de esta decisión y de los patrones de diseño adoptados.

---

[3] Una cuenta NICE es una cuenta que permite emplear los diversos servicios proporcionados por el CERN y permite la autenticación en el dominio. Normalmente son cuentas asociadas a una persona y a un correo electrónico, pero puede haber cuentas NICE de servicio, por ejemplo para aplicaciones.

3.3.2 Los principios ISP y el OCP

En el diseño de la aplicación se ha tenido presente la aplicación del principio ISP (acrónimo de Inteface Segregation Principle, *principio de segregación por interfaces*) que dice que:

Una clase A que tiene una dependencia con una clase B no debe estar obligada a depender de los métodos de dicha clase que no vaya a utilizar.

La idea detrás de este principio es siempre programar contra las interfaces de las clases y nunca contra sus implementaciones. Esto cobra especial sentido si por ejemplo se requiere cambiar el sistema de persistencia. Si se hubiese programado directamente contra la implementación, sería necesario modificar una gran parte de la aplicación (capa de datos y capa de servicios). Sin embargo si se programa contra la interfaz, se podría cambiar de implementación con facilidad, evitando tener que modificar todo el código, lo que violaría el principio OCP (open/closed principle, *principio abierto/cerrado*) que reza:

El código desarrollado debe estar cerrado a las modificaciones y abierto a la extensibilidad.

De esta manera, se podría extender las funcionalidades del código añadiendo una nueva implementación, sin necesidad de cambiar el código existente.

3.3.3 El principio DRY y los patrones de diseño DAO y GenericDAO

El patrón DAO (Data Access Object) se aplica cuando se necesita separar las entidades de las funcionalidades de persistencia de las clases. Esto quiere decir que una clase entidad (son aquellas que encapsularán los datos contenidos en una tabla de la base de datos), no debe contener la lógica de persistencia (métodos de grabación, consulta, borrado y actualización en la base de datos).

Por tanto, cada DAO está relacionado directamente con una entidad. En el DAO se implementarán las operaciones de persistencia de datos y la entidad tan solo efectuará labores de "contenedor" de información.

El patrón DAO tiene el aspecto de la Figura 17:

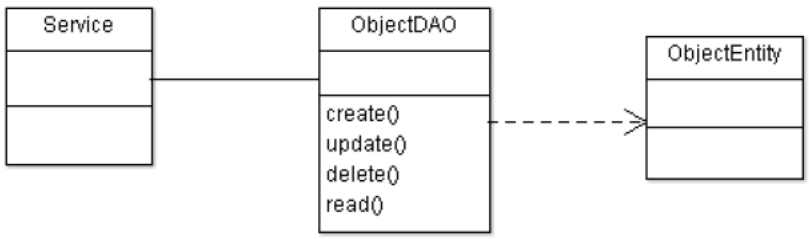

Figura 17: El patrón DAO

Pero debido a que (por ejemplo) puede suceder que se cambie en un futuro el sistema de gestión de bases de datos, no es una buena práctica programar contra la implementación de los DAOs (ni de ninguna clase en general) sino hacerlo contra sus interfaces (principio ISP), permitiendo la extensibilidad del código llegado el caso (principio OCP).

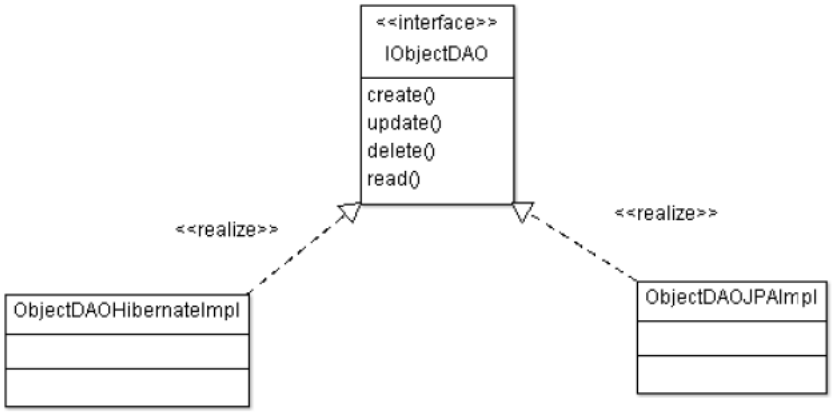

Figura 18: Programación contra interfaces

En la Figura 18 se puede observar cómo se define una interfaz que define los métodos de persistencia que cada una de las diferentes implementaciones implementará. Si se quiere cambiar de un sistema de persistencia como Hibernate a otro como JPA, no hará falta modificar las clases, sino crear una nueva implementación.

Pero existe un gran problema a la hora de crear los DAOs. Estos DAOs requieren una gran cantidad de código repetido que será básicamente el mismo siempre para cada entidad. Los métodos de persistencia tendrán el mismo aspecto en todos los DAOs, lo que lleva a una gran repetición de código y que

viola el principio DRY (Don't repeat yourself, *no te repitas a ti mismo*) y que se puede enunciar:

Ninguna pieza de información debe repetirse debido a que la replicación incrementa la dificultad en posteriores cambios y evoluciones, lo que podría comprometer la claridad del código y abrir un espacio para inconsistencias

Para solucionar este problema se empleará el patrón Generic DAO. EL patrón Generic DAO define una nueva interfaz que empleará genéricos para parametrizar las entidades y las claves primarias de las tablas. De esta manera, la implementación solo se encontrará en una sola clase, omitiendo la repetición de código.

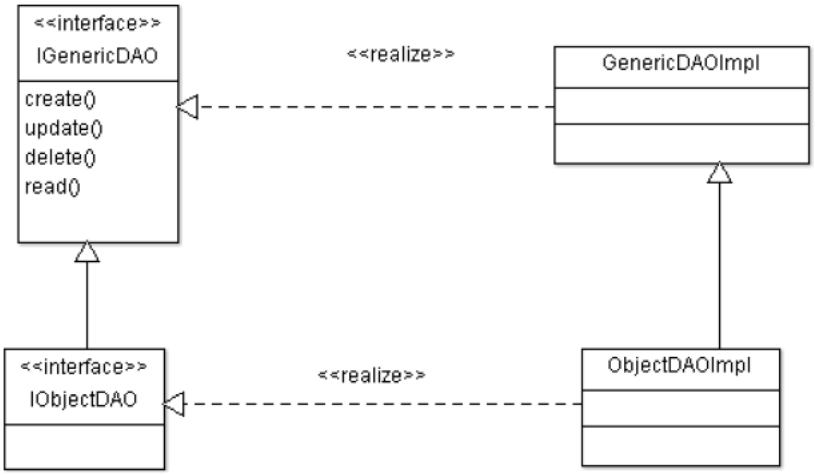

Figura 19: Patrón Generic DAO

Si se echa un vistazo al diagrama de clases de la Figura 19, se puede observar cómo la interface DAO hereda de la interface GenericDAO y la implementación hereda de la implementación GenericDAOImpl. Mediante esta arquitectura y el uso de genéricos, los DAOs para las tablas automáticamente heredarán los métodos básicos debidamente parametrizados gracias al uso de genéricos. En cada una de las implementaciones específicas para los DAO, se pueden añadir los métodos para realizar consultas más "a medida" a las tablas, que se salgan un poco de las típicas operaciones definidas en el Generic DAO.

3.3.4 La capa de servicios de acceso a datos

Sobre la capa de datos se situará la capa de negocio, donde la mayor parte de la lógica de negocio se llevará a cabo. En una SOA esta capa se corresponde con la capa de servicios y en esta capa de servicios se pueden distinguir normalmente tres capas, como ya se había comentado en el capítulo anterior.

La capa de servicios de acceso a datos actúa de alguna manera como *patrón façade* y su función será desacoplar completamente ambas capas. Existirá un servicio de acceso a datos por cada DAO, de manera que cuando un servicio de negocio o de integración necesita acceder a la base de datos, **lo hará a través del servicio de acceso a datos correspondiente** y nunca a través del DAO.

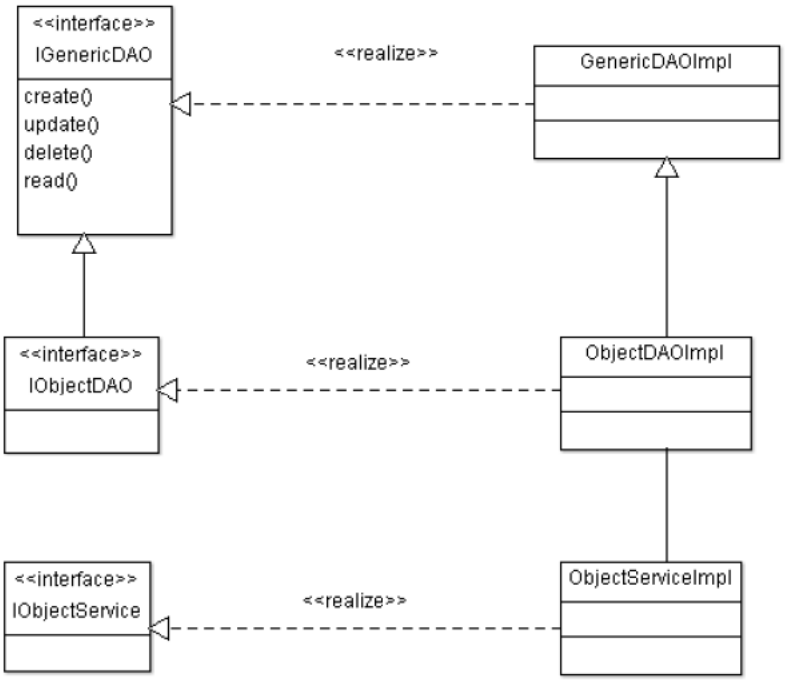

Figura 20: Capa de servicios de acceso a datos

Se puede observar en el diagrama de clases de la Figura 20 cómo los servicios también serán programados contra interfaces, pudiendo tener de esta manera diferentes implementaciones.

3.3.5 La capa de presentación y Java Server Faces

De la capa de presentación se encargará el framework Java Server Faces (JSF). Es una tecnología para aplicaciones Java web que simplificará el desarrollo de interfaces de usuario. JSF incluye un conjunto de APIs para representar componentes de la interfaz y administrarlas, etiquetas personalizadas para JSP o beans administrados

JSF se basa en el patrón de diseño MVC (Modelo-Vista-Controlador), que se puede ver en la Figura 21 y emplea tres módulos separados:

1. El **modelo** que contiene los datos
2. La **vista** que muestra la interfaz de usuario
3. El **controlador** que maneja el proceso de la aplicación

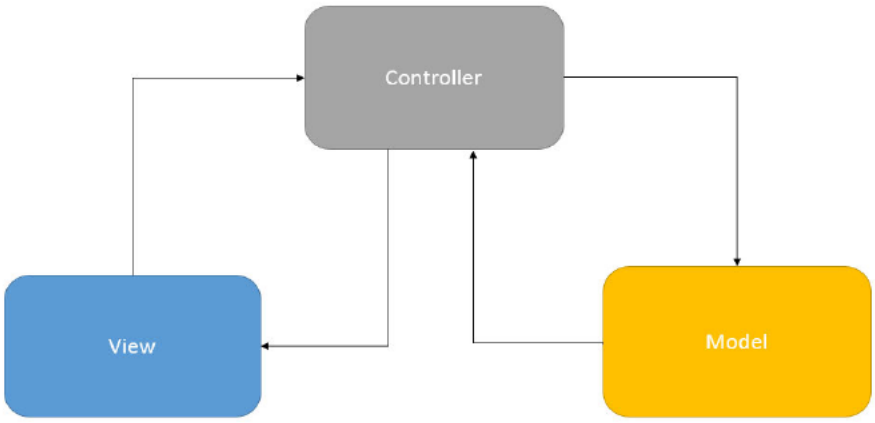

Figura 21: Modelo-Vista-Controlador

JSF proporciona una serie de etiquetas HTML extendidas proporcionadas para la parte de la Vista (las páginas xhtml), un Controlador llamado Faces Servlet y unas clases Java llamadas ManagedBeans donde se encuentra el Modelo. En estos ManagedBeans se podrá acceder a los diferentes servicios definidos en la arquitectura de la aplicación mediante inyección de dependencias, como se comentará en el apartado siguiente.

3.3.6 La inversión del control (IoC)

La inversión del control es un principio de desarrollo que procura liberar al programador de la instanciación de dependencias y en general del uso del operador new. El principio dice que:

La responsabilidad de la instanciación de los objetos no debería recaer en los programadores sino que otra clase debería ocuparse de ello.

Para realizar la inversión del control existen numerosas estrategias, pero en este caso se empleará la inyección de dependencias (DI, dependency injection), de lo que se encargará el framework Spring [2].

La idea detrás de esta arquitectura que se ha construido es que Spring instanciará por él mismo los DAO y los servicios e inyectará las dependencias necesarias en las clases marcadas con las anotaciones correspondientes, de manera que el programador no tendrá que crear sus propias instancias, sino que se inyectarán aquellas creadas por Spring y guardadas en su contenedor.

3.3.7 Generación automática del código de las capas de persistencia y servicios

3.3.7.1 Introducción

La separación en diferentes capas tradicional que es habitual en arquitecturas orientadas a servicios (SOA) y que se ha explicado anteriormente, trae ligadas muchas ventajas como el bajo acoplamiento entre las distintas capas y por lo tanto una mayor escalabilidad de la aplicación. Pero recrear esta arquitectura puede ser un trabajo muy repetitivo debido a la gran cantidad de clases similares que deben crearse.

Cada tabla de la base de datos a la que va a accederse tendrá un DAO, una entidad y un servicio, además de cada una de sus interfaces al aplicar el principio ISP y la base de datos puede tener docenas de clases y vistas.

Con una sencilla multiplicación puede hacerse una idea de la gran cantidad de código repetitivo que habrá de crearse.

Este apartado pretende ser un pequeño alto en el camino donde se explicarán brevemente las técnicas y estrategias de generación automática de código que se han empleado en este proyecto y que han ayudado a reducir notablemente el

tiempo empleado en la programación de las diferentes clases de persistencia y de servicios de acceso a datos, con el consiguiente ahorro de tiempo y de esfuerzo.

3.3.7.2 Identificación de la línea de productos

Para generar el código, se empleará la programación generativa (GP) que según [3], *es un paradigma de ingeniería del software basado en el desarrollo de familias de sistemas. El producto final de las GP es un modelo generativo capaz de sintetizar programas de una familia a partir de especificaciones de alto nivel de abstracción.*

Así pues el objetivo principal será la generación automática de código para los DAOs, las interfaces de los DAOs, los servicios de acceso a datos, sus interfaces y eventualmente se podrían generar también las entidades, aunque para esto existen herramientas como Hibernate tools que ya realizan ese trabajo.

Por tanto serán cuatro clases por tabla de la base de datos, que dependiendo del número de tablas de la misma, llevaría muchísimo tiempo de programación manual (con los consiguientes riesgos de errores de programación, incrementados en tareas aburridas y repetitivas).

Se podrán considerar estos ítems a generar como una línea de productos y su variabilidad vendrá dada fundamentalmente por la base de datos, incluyendo:

1. Nombres de las tablas
2. Claves primarias
3. Campos de las tablas
4. Tipos de datos

Los nombres de las tablas serán necesarios para generar los nombres de los ficheros (clases e interfaces a crear). Dado que se homogenizará el nombre de estos elementos, solo el nombre de la tabla será necesario.

Las claves primarias serán necesarias para generar los DAO ya que éstos están parametrizados por genéricos y esta parametrización consiste en el nombre de la clase entidad y su clave primaria.

Los campos de las tablas y los tipos de datos solo serán necesarios para generar las entidades (si se quisieran generar mediante un generador de código y no mediante otras herramientas). En este caso es importante subrayar que los tipos deberán ser mapeados de Oracle a Java[4].

[4] Puede encontrarse las conversiones más adecuadas de distintas bases de datos a Java en el siguiente enlace: https://docs.oracle.com/cd/E19501-01/819-3659/gcmaz/

En el siguiente cuadro de texto se puede observar la interface creada para una de las tablas de la base de datos, a modo de ejemplo.

Como se puede observar, la interface hereda de la interface IGenericDAO como se puede ver en el Listado 3. Por otro lado no se definen las signaturas de los métodos ya que vendrán datos directamente por la interfaz IGenericDAO.

```
public interface IExampleEntityDAO extends
IGenericDAO<ExampleEntity, ExampleEntityId>{

}
```

Listado 3: Ejemplo de interfaz que hereda del DAO Genérico

Echando un vistazo a las partes variables de la interfaz, el código será siempre el mismo salvo para el nombre de la entidad y su clave primaria (ambos datos proporcionados por la base de datos).

De manera análoga se podrán tratar tanto la implementación como la interface e implementación del servicio de acceso a datos.

3.3.7.3 Definición de un lenguaje específico de dominio (DSL)

Se ha de diseñar un lenguaje específico de dominio (DSL, *Domain Specific Language*) para implementar la variabilidad de la línea de productos, que sirva como input para alimentar el generador de código.

Por simplicidad, se ha escogido un DSL basado en XML, haciendo que resulte más sencilla su lectura por el generador. En el DSL se especificarán las estructuras de las tablas en la base de datos y tendrá un aspecto similar al del Listado 4:

```
<tables>
<table name="web_datapoint">
<field name="datapoint_id" datatype="number(2,3)" pk="true"/>
<field name="datapoint_description" datatype="varchar2(64 char)"/>
   <field name="datapoint_type" datatype="varchar2(1 char)"/>
   <field name="datapoint_feeder_id" datatype="number" />
   <field name="datapoint_mult" datatype="number"/>
</table>
</tables>
```

Listado 4: Ejemplo de DSL

El propio XML[5] es muy auto explicativo. En cada nodo <table> existirán varios campos dados por los nodos <field>. Solo uno de ellos albergará el atributo clave primaria a true. Los tipos de datos (datatypes) serán los mismos de Oracle.

3.3.7.4 El generador de código

Para llevar a cabo la generación, se ha desarrollado un pequeño generado en Ruby, aprovechando las características de este lenguaje. Se empleará la librería rexml para leer el archivo XML con la definición de las tablas a generar y se emplearán plantillas para la generación. La librería para trabajar con las plantillas ha sido erb. Existirán cinco tipos de plantillas diferentes, una para cada elemento a generar si se incluyen las entidades.

El generador de la Figura 22 puede considerarse un generador con configuración como bien se explican en [4] ya que recibe como input un XML que leerá y almacenará en memoria, creando una jerarquía de clases que encapsulará la información contenida en el mismo. Después, una clase transformadora se encargará empleando erb de generar la salida, basada en la jerarquía de clases guardada en memoria. Estas clases serán la clase Database, la clase Table y la clase Field: exactamente la misma información que se guarda en el XML.

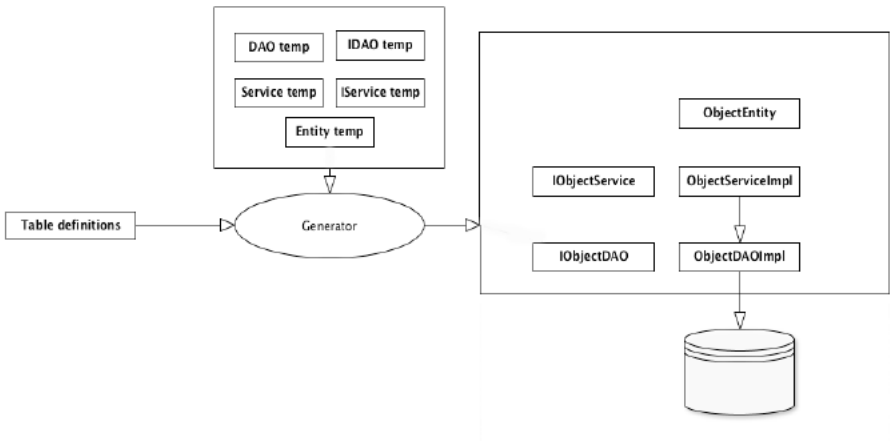

Figura 22: El generador de código

[5] Se puede encontrar más información sobre lenguajes de específicos de dominio en [3], apartado 3.2.2.1 *Especificación de un DSL*

En la salida estarán los cinco archivos por tabla generado (los DAO, sus interfaces, los servicios y sus interfaces y la entidad, si se desea generar).

La generación de estas clases garantiza la automatización de la creación de las capas de persistencia y de servicios de acceso a datos, garantizando las operaciones de persistencia básicas definidas por el DAO Genérico.

Obviamente, si se desean otras operaciones más complejas, se podrán implementar directamente en los DAOs ya generados, extendiendo su funcionalidad.

3.3.7.5 Conclusiones

En este apartado se ha querido mostrar cómo aplicando técnicas de generación de código se ha podido ahorrar mucho tiempo y esfuerzo en la realización de tareas repetitivas, incluso debiendo de crear el archivo XML con la definición de las tablas.

También se ha creído interesante exponer en este capítulo el generador de código realizado, a pesar de ser producto alternativo del proyecto WebEnergy, pero que es digno de mención por su interés práctico y académico.

4 Conclusiones y trabajo futuro

4.1 Resultados

4.1.1 Introducción

A día de hoy, aunque el proyecto sigue en evolución, se ha cumplido con los requisitos funcionales propuestos y la aplicación ha sido desplegada en producción. Especialmente útil ha sido la monitorización del gasto energético, que es la base del proyecto.

Por otro lado gracias a los conocimientos adquiridos en el máster se ha conseguido realizar un diseño mantenible que minimice, en la medida de lo posible, los cambios futuros.

En este capítulo se van a presentar los resultados obtenidos por la aplicación respecto a los consumos reales proporcionados por el proveedor de energía en la factura. Hay que tener en cuenta que el valor proporcionado por éste será el valor total (consumo total) que se ha consumido por todo el CERN.

Se tratará de esta manera de ver la fiabilidad de los datos proporcionados a lo largo del año 2015 por WebEnergy

4.1.2 Consumo real vs. Consumo estimado por WebEnergy

La primera comparación a realizar será ver cuán aproximados son los valores calculados por WebEnergy a los proporcionados por el proveedor eléctrico. Dado que solo se cuenta con un dato global, no será posible realizar una comparación a nivel de Consumer / Super Consumer sino que se comparará con el dato total de consumo para el nodo raíz o Super Consumer CERN.

Hay que recordar que el valor dado por el proveedor es un dato a posteriori y por lo tanto proporcionado cuando se genera la factura. Si los datos de WebEnergy son correctos, se podrá adelantar el importe de esta factura, y se sabrá por tanto el importe a pagar con antelación.

Por otro lado hay que tener en cuenta que el valor calculado para el Super Consumer CERN es la culminación de los cálculos para toda la arquitectura, por lo que habrá que suponer que si este cálculo es correcto, también lo será para todos sus subconsumers.

En la Tabla 1 del anexo, se recopilan los valores de consumos desgranados por días comparando aquellos aportados por el proveedor en la factura (invoice) con los obtenidos mediante la aplicación.

En la última columna se muestra la diferencia entre ambos valores en porcentaje, teniendo en cuenta que:

Diferencia < 0 % → Sobreestimación sobre el consumo real

Diferencia > 0 % → Subestimación sobre el consumo real

O sea, si es menor que cero, WebEnergy estará estimando un valor superior al real y viceversa.

Y en la Figura 23 los datos de los consumos presentados en forma gráfica (consumo en kWh / día):

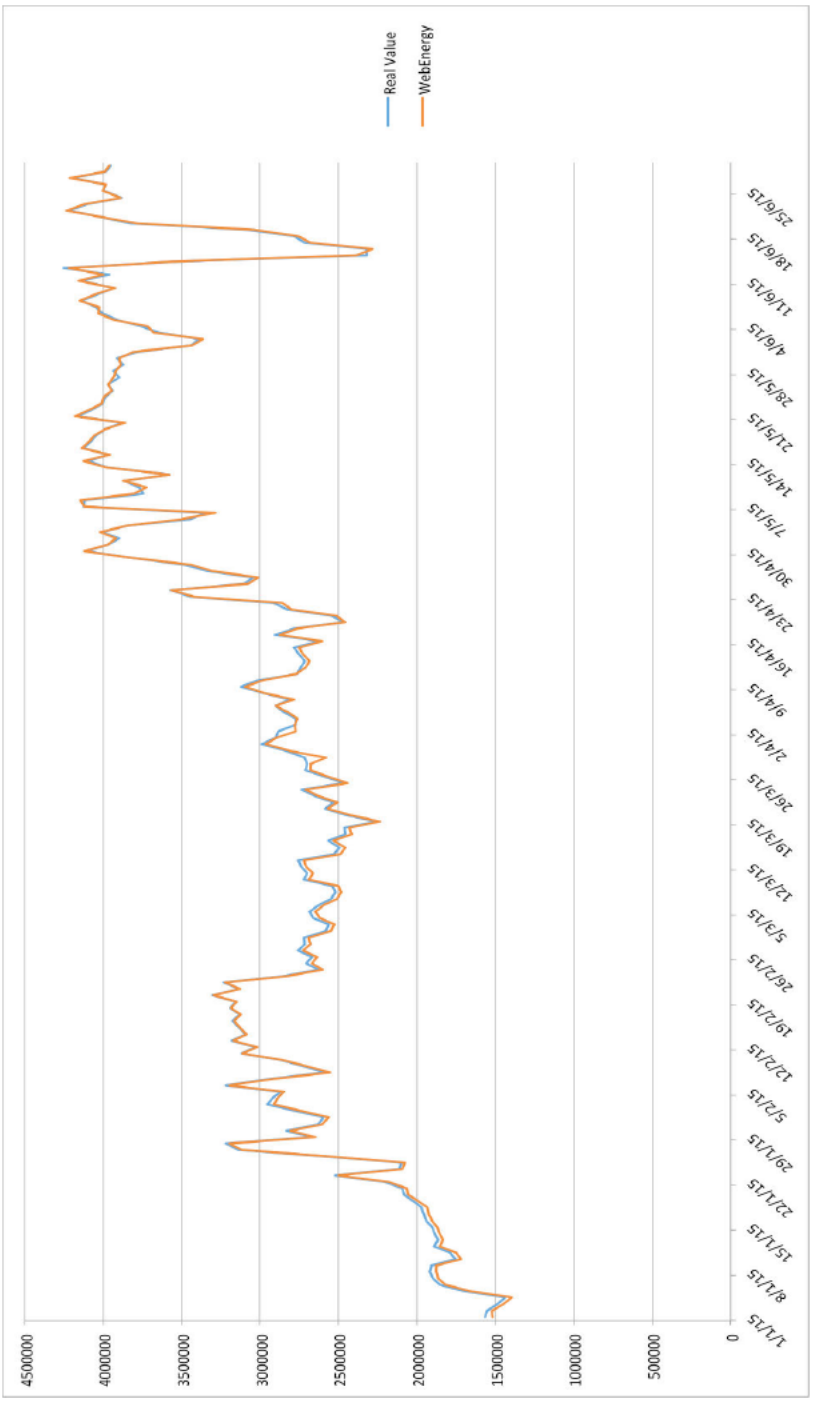

Figura 23: Consumo real vs. Consumo estimado

4.1.3 Error medio y desviación típica

Como se ve en la imagen anterior, los resultados en la gráfica son muy próximos. Se observa una diferencia máxima en un pico de un **4,87%** y la diferencia mínima se sitúa en el **0,005%**

La diferencia media o error medio (siempre que se consideren los datos dados por el proveedor como buenos) se situará en el **0,726%**

Este dato es importante ya que indica que se han logrado calcular los consumos con un error inferior al 1% y por lo tanto hacen muy fiables los cálculos, por lo que se puede concluir que también la segregación de consumos por Consumers será muy aproximada al consumo real de los mismos aunque se carezca de ese dato para poder comparar.

Por otro lado, la desviación típica de la muestra, se sitúa en el **0,953%**

El gráfico de la Figura 24 muestra la evolución en porcentaje del error diario respecto al cero (valor ideal), la diferencia media y la desviación típica. Se puede observar cómo en la parte final, los valores se acumulan simétricamente en torno al cero, por lo que es presumible que la tendencia sea a mejorar estos valores.

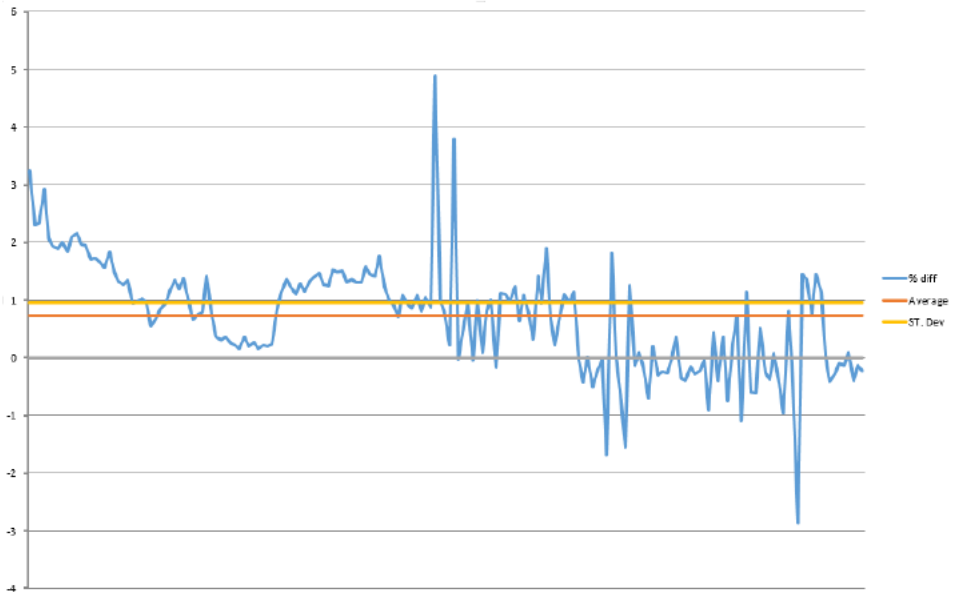

Figura 24: Error en % del consumo estimado respecto al valor real

4.2 Trabajo futuro

4.2.1 Introducción

Son múltiples las aplicaciones futuras y posibilidades de expansión que tiene la aplicación. En algunas ya está proyectada su realización y otras se está estudiando la posibilidad de hacerlas en un futuro. Las comentadas en este apartado son algunas de las más interesantes.

4.2.2 Aumento de la resolución

Uno de los puntos interesantes a la hora de ampliar las funcionalidades de la aplicación es aumentar la resolución de los datos publicados. Hasta ahora, estos datos se generan y publican diariamente –esto es, un dato por día– por lo que no es posible saber la evolución de los consumos en intervalos inferiores a una jornada.

Esto viene dado fundamentalmente por una limitación en la publicación de los datos a la base de datos LHC Logging por el SCADA. Hay que recordar que estos datos se publican una vez al día a media noche y luego WebEnergy los recupera para realizar el cálculo de consumos.

Este intervalo de publicación y posterior cálculo podría reducirse por ejemplo, a cada hora. En teoría así se podrían tener datos de consumo cada hora, y ver su evolución a lo largo del día.

En la práctica esto es más complicado debido a temas de rendimiento en la parte del SCADA: se obligaría a éste a publicar los datos cada hora durante las 24 horas, incluidos los momentos del día de mayor actividad. Además requeriría cambios en la aplicación tanto en la parte de la interfaz como en el cálculo del árbol de consumos.

4.2.3 Creación de nuevos árboles

Otra opción de ampliación de las funcionalidades de WebEnergy sería la creación de nuevos árboles de Consumers. Esto quiere decir el contar con nuevas agrupaciones de Consumers alternativas a las del árbol principal.

Uno de estos árboles alternativos que más interés despierta dentro del grupo EN-EL, es el de agrupar Consumers por subestaciones eléctricas, de manera que

se pudiera saber en todo momento cuáles serían los consumos de dichas subestaciones.

4.2.4 Mapa de energía

Este punto viene directamente relacionado con el anterior. Las subestaciones eléctricas se pueden asociar con un área geográfica a la que proveen de energía. Si se tuviese el árbol de Consumers por subestaciones eléctricas, se tendrían también, de alguna manera, los consumos energéticos por áreas dentro del CERN.

La idea detrás sería representar estos consumos de manera gráfica en un *mapa de consumos*. Se podría así representar mediante un código de colores los diversos edificios en función de sus consumos, añadiendo el componente geográfico a los datos.

4.2.5 Ampliación de la API REST para publicación de datos

La aplicación es accedida diariamente desde los diferentes departamentos del CERN para recuperar los datos de consumos de los diferentes experimentos o infraestructuras en las que estén interesados.

Sería interesante y probablemente útil para estas personas el poder acceder a los datos proporcionados por WebEnergy de manera programática, para insertarlos en sus cálculos o aplicaciones propias.

La manera más accesible para realizar esto sería ampliando la API REST de la aplicación para publicar los datos. De esta manera, mediante una simple petición http GET desde cualquier software, se podrían recuperar los datos requeridos en formato XML o JSON y podrían así ser directamente leídos e insertados en las aplicaciones sin importar en qué leguaje estén programadas y sin necesidad de crear conexiones a ninguna base de datos.

4.2.6 Cálculo de la potencia

Otro de los puntos de posible trabajo futuro es el de realizar de manera análoga a la energía, el cálculo de la potencia. Actualmente se ofrece el cálculo de la misma, pero a partir del consumo de energía (al ser MWh y dividirlo por 24, se obtiene la potencia diaria en MW). La idea sería realizar el cálculo de la potencia a través de otros datapoints en el archivo con sufijos *PM10*, *P+* y *Q+*.

4.2.7 Exportación a otros entornos industriales

Una de las cuestiones más interesantes del proyecto es la posibilidad de exportar la idea a otros entornos industriales similares, que cuenten con un sistema SCADA. Obviamente esta exportación no será trivial y requerirá trabajo de configuración y de desarrollo, pero será posible realizar una aproximación a la solución presentada en este texto que pueda conseguir resultados similares.

5 Referencias

[1] T. Erl, Service oriented architecture, a field, Prentice Hall, 2009.

[2] C. Walls, Spring in Action, 4th edition, Manning, 2015.

[3] R. H. Gil, Metodología de desarrollo de software basada en el paradigma generativo. Realización mediante la transformación de ejemplares, UNED, 2007.

[4] J. Herrington, Code Generation in Action, Manning, 2003.

[5] C. Szyperski, Component Software, Addison-Wesley, 2002.

[6] R. Day, How to Write and Publish a Scientific Paper, 4th edition, Cambridge University Press, 1995.

[7] S. Metsker y W. wake, Design Patterns in Java, Addison-Wesley, 2006.

Anexo: tabla de consumos

Date	Invoice (kWh)	WebEnergy (kWh)	% Diff
1/1/15	1565900	1515125	3,242544224
2/1/15	1555500	1519644	2,305110897
3/1/15	1490933,333	1456064	2,338758719
4/1/15	1437483,333	1395308	2,933970249
5/1/15	1693983,333	1658625	2,087289328
6/1/15	1854883,333	1819066	1,930974994
7/1/15	1898716,667	1862859	1,888521194
8/1/15	1915800	1877344	2,007307652
9/1/15	1909166,667	1874178	1,832666958
10/1/15	1752333,333	1715406	2,107323569
11/1/15	1784966,667	1746588	2,150105511
12/1/15	1889700	1852458	1,970789014
13/1/15	1867616,667	1831276	1,94583114
14/1/15	1887116,667	1854754	1,714926652
15/1/15	1903383,333	1870304	1,737922822
16/1/15	1936666,667	1904695	1,650860585
17/1/15	1952916,667	1922203	1,572707489
18/1/15	1968700	1932414	1,843145223
19/1/15	2021850	1991532	1,499517768
20/1/15	2082500	2054898	1,32542617
21/1/15	2090050	2063663	1,262505682

22/1/15	2209400	2179930	1,333846293
23/1/15	2520566,667	2497058	0,93267387
24/1/15	2112533,333	2092108	0,966864428
25/1/15	2099950	2078368	1,027738756
26/1/15	2584164	2559968	0,936318283
27/1/15	3136486	3119111	0,553963895
28/1/15	3215727,333	3194161	0,670651803
29/1/15	2668266,667	2645940	0,836747951
30/1/15	2829652	2802941	0,943967668
31/1/15	2633566,667	2603042	1,159061855
1/2/15	2596200	2561275	1,345235344
2/2/15	2771166,667	2737892	1,200745775
3/2/15	2951633,333	2911038	1,3753515
4/2/15	2919966,667	2891590	0,971814747
5/2/15	2869328,667	2850194	0,666869114
6/2/15	3219235,333	3195173	0,747454934
7/2/15	2942643,333	2919269	0,794331174
8/2/15	2591100	2554338	1,418779669
9/2/15	2721033,333	2701190	0,729257267
10/2/15	2872133,333	2861494	0,370433128
11/2/15	3119049,333	3109337	0,311387615
12/2/15	3027068,667	3015750	0,373915095
13/2/15	3180631,333	3172293	0,262159693
14/2/15	3090924	3084295	0,214466613
15/2/15	3130254	3125004	0,167718019
16/2/15	3176660,667	3165262	0,358825442
17/2/15	3129016,667	3122957	0,193660415

18/2/15	3192184	3183451	0,273574456
19/2/15	3154853,333	3149611	0,16616726
20/2/15	3303798,667	3296531	0,219979103
21/2/15	3134488,667	3128441	0,192939497
22/2/15	3232648	3224979	0,237235851
23/2/15	2854912	2830630	0,850534097
24/2/15	2626633,333	2596877	1,132869706
25/2/15	2703950	2667354	1,353427393
26/2/15	2670350	2637570	1,22755444
27/2/15	2758183,333	2727854	1,099612668
28/2/15	2715333,333	2680423	1,28567395
1/3/15	2722183,333	2690687	1,157024692
2/3/15	2579966,667	2545819	1,323570072
3/3/15	2563433,333	2527825	1,389087552
4/3/15	2662000	2622722	1,475507137
5/3/15	2683566,667	2649354	1,274895351
6/3/15	2625500	2592522	1,256065511
7/3/15	2549083,333	2510065	1,530680964
8/3/15	2521483,333	2484064	1,484020649
9/3/15	2535733,333	2497675	1,500880745
10/3/15	2723366,667	2687826	1,305026866
11/3/15	2701266,667	2664336	1,367161085
12/3/15	2738066,667	2702550	1,29714397
13/3/15	2757700	2721447	1,314610001
14/3/15	2530466,667	2490590	1,57586216
15/3/15	2491516,667	2455465	1,446976741
16/3/15	2564950	2528828	1,408292559

17/3/15	2457550	2414274	1,760940774
18/3/15	2462035,333	2431617	1,235495402
19/3/15	2255200	2232605	1,001906705
20/3/15	2449050	2427034	0,898960822
21/3/15	2586150	2567439	0,723507917
22/3/15	2531466,667	2504125	1,080072158
23/3/15	2653166,667	2628824	0,917494818
24/3/15	2736266,667	2712735	0,859991716
25/3/15	2468566,667	2441847	1,082395992
26/3/15	2585766,667	2564810	0,810462403
27/3/15	2708583,333	2680301	1,044174384
28/3/15	2702033,333	2678058	0,88730709
29/3/15	2713066,667	2580877	4,87233389
30/3/15	2827750	2800310	0,970382813
31/3/15	2987870,667	2964540	0,780845936
1/4/15	2902800,667	2896702	0,210095951
2/4/15	2884551,333	2775263	3,788746349
3/4/15	2776000,667	2776423	-0,015213733
4/4/15	2775683,333	2760777	0,537032923
5/4/15	2846166,667	2819678	0,930678691
6/4/15	2898600	2899837	-0,042675775
7/4/15	2810117,333	2782236	0,992176839
8/4/15	2977526	2974698	0,09497818
9/4/15	3122402	3097214	0,806686647
10/4/15	3012238	2981519	1,019806536
11/4/15	2762918,667	2767641	-0,17091829
12/4/15	2742783,333	2712138	1,117307844

13/4/15	2714616,667	2684494	1,109647157
14/4/15	2751917,333	2724904	0,981618634
15/4/15	2785667,333	2751460	1,227976253
16/4/15	2623867,333	2607059	0,640593871
17/4/15	2903502,667	2871915	1,087915883
18/4/15	2786004,667	2764323	0,778235117
19/4/15	2464033,333	2455866	0,331461966
20/4/15	2548650	2512573	1,415533714
21/4/15	2827633,333	2800984	0,942460715
22/4/15	2914734	2859361	1,899761694
23/4/15	3443892,667	3420734	0,67245611
24/4/15	3571305,333	3563602	0,215700776
25/4/15	3099306,667	3076655	0,730862386
26/4/15	3044868	3011045	1,110819911
27/4/15	3340359,333	3308648	0,949338983
28/4/15	3481529,333	3441548	1,148384216
29/4/15	3833375,333	3834982	-0,041912584
30/4/15	4102330,667	4119789	-0,42557109
1/5/15	3972408	3971874	0,013442728
2/5/15	3898802,667	3918358	-0,501572791
3/5/15	4008726	4018319	-0,239302961
4/5/15	3843821,333	3845321	-0,039014994
5/5/15	3444842	3502716	-1,680018997
6/5/15	3347370,667	3286327	1,823630328
7/5/15	4124282,667	4122668	0,039150243
8/5/15	4122783,333	4146641	-0,578678643
9/5/15	3742972	3801224	-1,556303387

10/5/15	3771480	3724255	1,252160955
11/5/15	3865660,667	3870797	-0,132870776
12/5/15	3577417,333	3574121	0,092142823
13/5/15	3971900,667	3977798	-0,148476355
14/5/15	4096081,333	4124929	-0,704274752
15/5/15	3965580,667	3957755	0,197339742
16/5/15	4125133,333	4137473	-0,299133765
17/5/15	4082629,333	4092176	-0,233836234
18/5/15	4044179,333	4054390	-0,252478088
19/5/15	3987844	3987622	0,005566918
20/5/15	3877178,667	3862955	0,366856106
21/5/15	4164458,667	4179104	-0,351674359
22/5/15	4080200,667	4096001	-0,387244026
23/5/15	4002762,667	4008411	-0,141110873
24/5/15	3986874,667	3997977	-0,278472093
25/5/15	3938910,667	3947967	-0,229919744
26/5/15	3965046,667	3966484	-0,036250099
27/5/15	3899995,333	3935587	-0,91260793
28/5/15	3934147,333	3917172	0,431486975
29/5/15	3872925,333	3887932	-0,387476271
30/5/15	3913845,333	3899623	0,363385165
31/5/15	3783225,333	3811083	-0,736347011
1/6/15	3441216	3432983	0,239246824
2/6/15	3389076	3363924	0,742149188
3/6/15	3637533,333	3676992	-1,084764401
4/6/15	3759727,333	3717074	1,134479433
5/6/15	3909074,667	3932428	-0,597413335

6/6/15	4005417,333	4029518	-0,601701762
7/6/15	4048022,667	4027202	0,514341652
8/6/15	4139041,333	4150036	-0,265633169
9/6/15	4035334	4050404	-0,37345112
10/6/15	3929473,333	3926866	0,066353252
11/6/15	4141908	4159653	-0,42842574
12/6/15	3959197,333	3996914	-0,95263417
13/6/15	4254702,667	4220071	0,813962088
14/6/15	3554518,667	3569432	-0,419559854
15/6/15	2321270,667	2387531	-2,854485446
16/6/15	2318133,333	2284306	1,459248821
17/6/15	2718033,333	2681113	1,358347334
18/6/15	2781988,667	2760410	0,775656167
19/6/15	3074694,667	3030216	1,446604346
20/6/15	3825220,667	3781331	1,147376073
21/6/15	4013862,667	4011809	0,051164348
22/6/15	4219012	4236361	-0,411210018
23/6/15	4106366,667	4118001	-0,283324269
24/6/15	3884197,333	3887629	-0,088349442
25/6/15	3998736	4003923	-0,12971599
26/6/15	3987150,667	3983664	0,087447577
27/6/15	4198114,667	4214283	-0,385133199
28/6/15	3984026,667	3989085	-0,126965348
29/6/15	3951054	3959537	-0,214702204

Tabla 1: Comparación de consumos estimado vs real