Proyecto Fin de Carrera

Diseñador BPM para el sistema IDbox

BPM designer for IDbox system

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Pérez Abascal, María Jesús
Director: González Harbour, Michael
Codirector: Sierra Sánchez, Miguel

Febrero - 2017

RESUMEN

Las nuevas tecnologías han provocado un enorme cambio en la forma de trabajar y de comunicarse en muchas áreas. Una de las áreas afectadas ha sido la relativa a los procesos industriales y la monitorización de éstos, ya que tradicionalmente, se realizaban desde lugares físicos concretos, mientras que, en la actualidad, ha surgido la necesidad de acceder a máquinas que se encuentran en un lugar geográfico distinto.

Este proyecto tiene por objetivo la creación de una solución web que permita el control y monitorización de los procesos industriales desde cualquier emplazamiento y dispositivo con soporte de navegadores web. En concreto, esta solución realiza parcialmente las funciones de una antigua aplicación de escritorio diseñada para el sistema SCADA IDbox, en particular aporta las funcionalidades de visualización y edición de diagramas para su motor de cálculo.

Palabras clave: SCADA, gráficos vectoriales, MVC, IDbox, cliente web

ABSTRACT

New technologies have brought a huge change in the ways of working and communicating in many areas. One of the affected areas has been industrial processes and their monitoring. Traditionally, control and monitoring were performed from specific physical locations, while, at present, there has been a need to access machines from different geographical places.

This project aims at creating a web solution that allows the control and monitoring of industrial processes from any location and device with support for web browsers. In particular, this solution partially performs the functions of an old desktop application designed for the IDbox SCADA system, In particular it provides the visualization and editing of diagrams for its business process management engine.

Keywords: SCADA, vector graphics, MVC, IDbox, web client

Facultad de Ciencias

ÍNDICE GENERAL

1 INTRODUCCIÓN 1
1.1 Introducción a los sistemas SCADA 1
1.2 Introducción a IDbox 3
1.3 Motivación 4
1.4 Objetivos 4
1.5 Estructura del documento 4
2 ESTADO DEL ARTE 6
2.1 IDbox 6
2.1.1 Arquitectura 6
2.1.2 Motor de cálculo (BPM) 8
2.1.3 Diagramas de cálculo 9
2.1.4 Base de datos de IDbox 10
2.2 Tecnologías empleadas 13
2.2.1 Modelo MVC 13
2.2.2 Arquitectura basada en capas 14
2.2.3 HTML5 15
2.2.4 ASP .NET MVC 19
3 HERRAMIENTAS Y METODOLOGÍA 21
3.1 Herramientas 21
3.1.1 Microsoft SQL Server 2012 21
3.1.2 Microsoft Visual Studio 2010 22
3.1.3 Internet Explorer 9 23
3.2 Metodología de desarrollo 24
4 DESARROLLO 26
4.1 Propuesta de arquitectura dentro del sistema 26
4.2 Captura y análisis de requisitos 27
4.3 Diagramas de secuencia 29
4.4 Diagrama de clases 31
4.5 Código y pruebas 33
4.5.1 Interacción 1: Visualización de un diagrama 33
4.5.2 Interacción 2: Consulta del valor en tiempo real (TR) de una puerta 38

5 CONCLUSIONES 42
6 LÍNEAS FUTURAS 43
7 BIBLIOGRAFÍA 44

Facultad de Ciencias

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Pirámide de automatización 1
Ilustración 2: Esquema de un sistema basado en SCADA 2
Ilustración 3: Componentes de IDbox a alto nivel 3
Ilustración 4: Esquema conceptual de IDbox 6
Ilustración 5: Detalle del componente de los servicios de análisis de IDbox 7
Ilustración 6: Vista de despliegue del cliente web de IDbox 8
Ilustración 7: Configurador del motor de cálculo de IDbox 9
Ilustración 8: Ejemplo de diagrama de cálculo de IDbox 10
Ilustración 9: Modelo de base de datos de IDbox 12
Ilustración 10: Esquema del patrón de diseño MVC 13
Ilustración 11: Esquema de la arquitectura basada en capas 14
Ilustración 12: Estructura básica de HTML 16
Ilustración 13: Ejemplo CSS 17
Ilustración 14: Ejemplo creado con Joint JS [12] 19
Ilustración 15: Interfaz del sistema de administración de base de datos SQL Server 2012 22
Ilustración 16: Interfaz del entorno de desarrollo Visual Studio 2010 23
Ilustración 17: Herramienta de desarrollo del navegador IE 24
Ilustración 18: Iteraciones llevadas a cabo en el desarrollo 25
Ilustración 22: Componentes del sistema IDbox 26
Ilustración 23: Diagrama de secuencia de visualización de un diagrama 30
Ilustración 24: Diagrama de secuencia de consulta del valor TR de una puerta 31
Ilustración 25: Diagrama de clases del sistema 32
Ilustración 26: Interacción 1: visualización de diagrama 37
Ilustración 27: Interacción 1: visualización de diagrama con vista previa 37
Ilustración 28: Interacción 2: visualización de valores en TR de puertas 41

Facultad de Ciencias

ÍNDICE DE TABLAS

Tabla 1: Etiquetas HTML 17
Tabla 2: Requisitos heredados del sistema padre 27
Tabla 3: Requisitos funcionales específicos del sistema 28
Tabla 4: Requisitos no funcionales específicos del sistema 29

1 INTRODUCCIÓN

Este capítulo recoge brevemente los antecedentes que motivan este Proyecto Fin de Carrera y describe los objetivos que ha de satisfacer.

1.1 INTRODUCCIÓN A LOS SISTEMAS SCADA

A lo largo de la historia, los procesos industriales han evolucionado enormemente, y, con ellos, la forma en la que se han controlado y gestionado. Desde métodos de control mecánicos y neumáticos hasta métodos eléctricos y la tecnología digital actual, el control de los procesos también ha evolucionado exponencialmente.

En la Ilustración 1, se muestra la pirámide de automatización, que representa los distintos niveles de control. La parte más baja de la pirámide está formada por los dispositivos físicos localizados en el proceso industrial, y a medida que se escala en la pirámide, se obtiene control, capacidad de gestión y de toma de decisiones, hasta llegar a la cumbre, donde se localizan los sistemas ERP, acrónimo de Enterprise Resource Planning (Planificación de Recursos Empresariales), que es software que permite la completa gestión de los recursos del proceso industrial.

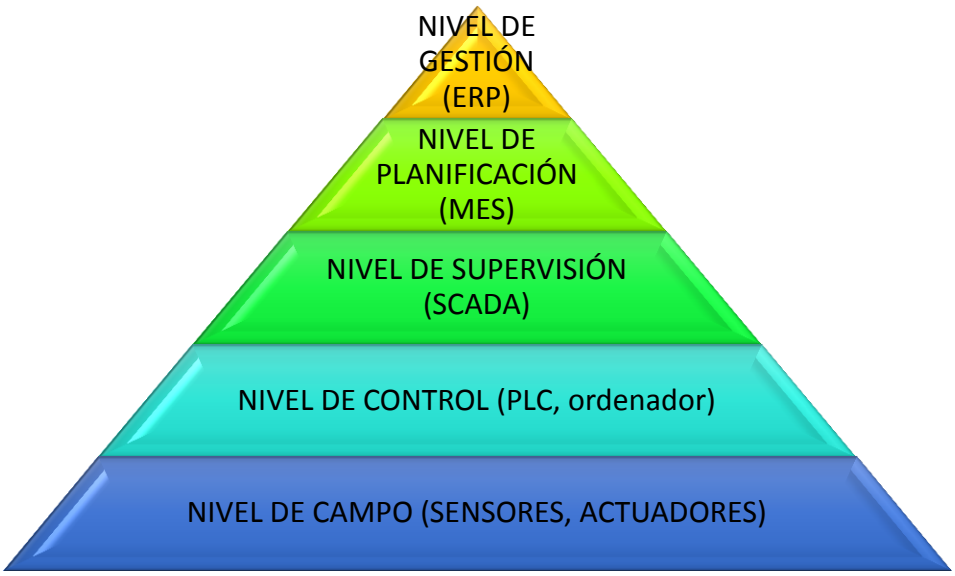

Ilustración 1: Pirámide de automatización

Se observa cómo en la parte más baja de la pirámide se encuentran los dispositivos de campo, es decir, los elementos físicos que están presentes en el proceso industrial: sensores y actuadores. En el siguiente nivel se sitúan los sistemas de control, como ordenadores y PLCs, acrónimo de Programmable Logic Controller (Controlador Lógico Programable) con los que se puede controlar el proceso industrial mediante la captura de señales de los sensores y la ejecución de órdenes mediante los actuadores. El tercer nivel corresponde a los sistemas de supervisión, control y adquisición de datos, conocidos comúnmente como SCADA, acrónimo de Supervisory Control And Data Acquisition (Supervisión, Control y Adquisición de Datos), que permiten el control de procesos distribuidos, además del almacenamiento de los datos captados. En el cuarto nivel se encuentran los sistemas de ejecución de producción, es decir, integran las funcionalidades de un SCADA con la información de gestión del proceso, como control de calidad, stocks y transportes, planificación de la producción, etc. Habitualmente, los sistemas MES, acrónimo de Manufacturing Execution System (Sistema de Ejecución de Manufactura) se hallan integrados en los sistemas ERP, el quinto y último nivel de la pirámide. Estos sistemas ofrecen la gestión total del proceso, añadiendo aspectos transversales a la automatización en sí como son la contabilidad, logística, facturación, recursos humanos, etc.

Un sistema SCADA [1] (nivel 3 de la Ilustración 1) permite controlar y supervisar procesos industriales a distancia. Facilita la realimentación en tiempo real con los dispositivos de campo (sensores y actuadores), y puede controlar el proceso automáticamente, a pesar de que, como su propio nombre indica, se trate de un sistema de supervisión, y por ello necesite un componente humano que tome las decisiones sobre las operaciones. Además, un sistema SCADA provee de toda la información que se genera en el proceso productivo (supervisión, control de producción, almacenamiento de datos, etc.) y permite su gestión e intervención.

A continuación, en la Ilustración 2, se muestra el esquema básico de un SCADA conectado a un proceso automatizado, que se corresponde con los tres primeros niveles de la Pirámide de automatización

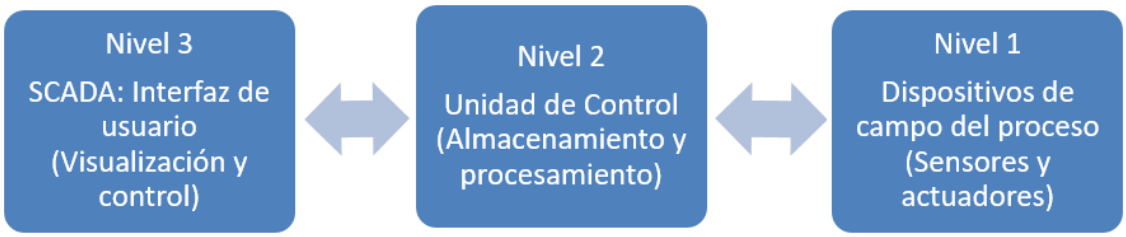

Ilustración 2: Esquema de un sistema basado en SCADA

Este tipo de sistemas de automatización es utilizado ampliamente en el control de infraestructuras críticas, que son los sistemas o servicios que son esenciales en el funcionamiento de la economía y la sociedad. Dado el uso cada vez más intensivo de las tecnologías de la información y comunicación, es vital la adaptación de estos sistemas a las necesidades actuales, tales como la posibilidad de acceder a la interfaz del sistema a distancia y desde dispositivos portátiles.

1.2 INTRODUCCIÓN A IDBOX

IDbox es un software de adquisición de información, integración, procesado y análisis, que desarrolla la compañía CIC Consulting Informático. Es decir, se trata de un sistema SCADA. Este software, como todos los sistemas SCADA, capta las señales y las almacena, de forma que sirven en la toma de decisiones durante la supervisión de un proceso industrial.

En la Ilustración 3 se muestran a alto nivel los componentes que forman IDbox.

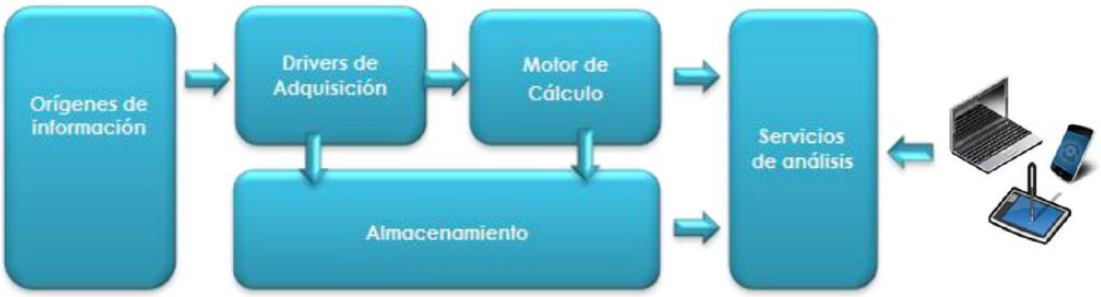

Ilustración 3: Componentes de IDbox a alto nivel

Uno de estos componentes es el motor de cálculo o BPM [2] (Business Process Management Engine, Gestión de Procesos de Negocio), que permite diseñar cálculos a partir de las señales adquiridas y posteriormente generar señales nuevas. Se denominan cálculos a los conjuntos de operaciones (aritméticas, lógicas…) que ejecuta el motor de cálculo. Los cálculos se configuran mediante una herramienta de escritorio clásica.

1.3 MOTIVACIÓN

En la actualidad, los sistemas SCADA son ampliamente utilizados en el control de infraestructuras críticas. Hasta ahora, su uso ha sido sujeto a lugares geográficos fijados, desde dónde se ha interactuado con el proceso industrial. Hoy en día, y tras la revolución tecnológica vivida, el concepto de control sobre el proceso industrial está cambiando, y con ello los propios sistemas que intervienen. Dado el uso cada vez más intensivo de las tecnologías de la información y comunicación, es vital la adaptación de estos sistemas a las necesidades actuales, tales como la posibilidad de acceder a la interfaz del sistema a distancia y desde dispositivos portátiles. Un método de acceso casi universal desde diferentes equipos deslocalizados geográficamente y con diferentes características es ofrecer interfaces basadas en servidores web y pensadas para ejecutarse en clientes soportados por navegadores web.

Es por este motivo que surge este proyecto final de carrera, en el que se propone la realización de un prototipo de diseñador BPM integrado en el componente web de IDbox.

1.4 OBJETIVOS

El objetivo de este proyecto fin de carrera es el diseño de una solución que integre el componente BPM Engine de IDbox en el cliente web y su implementación. Para ello, se desglosa en los siguientes objetivos específicos:

- Análisis del componente BPM en la herramienta IDbox
- Propuesta de diseño y arquitectura en el cliente web
- Estudio de las tecnologías disponibles para realizar el desarrollo
- Desarrollo de la integración en el cliente web
- Realización de pruebas

1.5 ESTRUCTURA DEL DOCUMENTO

El objetivo de este documento es explicar detalladamente este proyecto y su desarrollo. Para ello, el documento se compone de siete capítulos estructurados de la siguiente forma:

El primer capítulo ha introducido al lector en el concepto de los procesos industriales y los sistemas SCADA, además de explicar brevemente la motivación y objetivos del proyecto.

El segundo capítulo detalla el estado del arte necesario para que el lector pueda comprender el desarrollo del proyecto.

El tercer capítulo recoge tanto las herramientas utilizadas en el desarrollo como la metodología seguida, de forma que tras este capítulo, el lector tenga ya una idea inicial de que cómo ha sido el proceso de desarrollo.

El cuarto capítulo explica con detalle el proceso de desarrollo. En él, se detallan los requisitos del proyecto, así como los diagramas que permiten entender la funcionalidad del mismo. Además, se incluyen muestras del código desarrollado y pruebas finales.

El quinto capítulo recoge las conclusiones a las que se ha llegado tras la finalización de este proyecto.

El sexto capítulo detalla las posibles líneas futuras de desarrollo encontradas al término del proyecto.

El séptimo y último capítulo recoge toda la bibliografía consultada y referenciada en este documento.

2 ESTADO DEL ARTE

Este capítulo recoge brevemente los conceptos y tecnologías necesarios para la correcta comprensión de este documento.

2.1 IDBOX

Tal y como se ha explicado en el capítulo *1.2Introducción a IDbox,* el presente proyecto se ha llevado a cabo sobre el SCADA IDbox. A continuación, se detalla un análisis de IDbox y del motor de cálculo, ya que es el componente sobre el que se centra el desarrollo del trabajo.

2.1.1 Arquitectura

IDbox está formado por una serie de componentes distribuidos que se comunican entre sí, tal y como se muestra en la Ilustración 4.

Ilustración 4: Esquema conceptual de IDbox

En primer lugar, el sistema dispone de drivers o conectores, que se comunican con los orígenes de información y son los encargados de comenzar el proceso de adquisición de datos (señales). En segundo lugar, los agentes son los encargados de etiquetar los datos procedentes de los conectores y de distribuirlos a los servicios de almacenamiento y al motor de cálculo (si se trata de una señal que interviene en un cálculo). El motor de cálculo permite diseñar cálculos a partir de las señales adquiridas

y generar nuevas señales. Se soportan operaciones aritméticas, avisos y notificaciones, alarmas, etc. Los servicios de almacenamiento se encargan de asegurar la persistencia de la información de forma cifrada y comprimida en la base de datos. Finalmente, los servicios de análisis obtienen datos tanto históricos como en tiempo real que muestran al usuario.

Tal y como se muestra en la Ilustración 5Ilustración 5, los servicios de análisis se desglosan en cuatro componentes: una capa de servicios internos, una capa de negocio y control, el cliente web y la API (*Application Program Interface* o Interfaz de Programación de Aplicación) de servicios externos. Los servicios internos son consumidos por la capa de negocio, que a su vez es usada por el cliente Web y por la API de servicios que se exponen a terceras aplicaciones. De esta forma los servicios internos son consumidos de forma controlada.

Ilustración 5: Detalle del componente de los servicios de análisis de IDbox

El cliente web de IDbox ha sido implementado con el patrón de diseño MVC (Modelo Vista Controlador). A su vez, el modelo se encuentra internamente implementado mediante una arquitectura de capas, concretamente dos: capa de acceso a datos (DAL: Data Access Layer) y capa de negocio (BL: Business Layer). En la implementación, se ha hecho uso de las siguientes tecnologías: .NET MVC, HTML5, CSS3, EXT.NET y JAVASCRIPT.

Por consiguiente, en el cliente web hay tres componentes claramente diferenciados: modelo, vista y controlador, además de las dos capas dentro del modelo. La vista se compone en casi su totalidad de código JavaScript, por lo que se ejecuta en el lado del terminal del usuario final. La comunicación con el cliente web se lleva a cabo a través del modelo con la capa de negocio y control. El modelo, concretamente la capa de acceso a datos, se comunica con el sistema de almacenamiento. La vista y el controlador se comunican con los dispositivos conectados al cliente web. La vista de despliegue del cliente web de la Ilustración 6 muestra la manera en la que interactúan estos componentes dentro de la arquitectura del sistema.

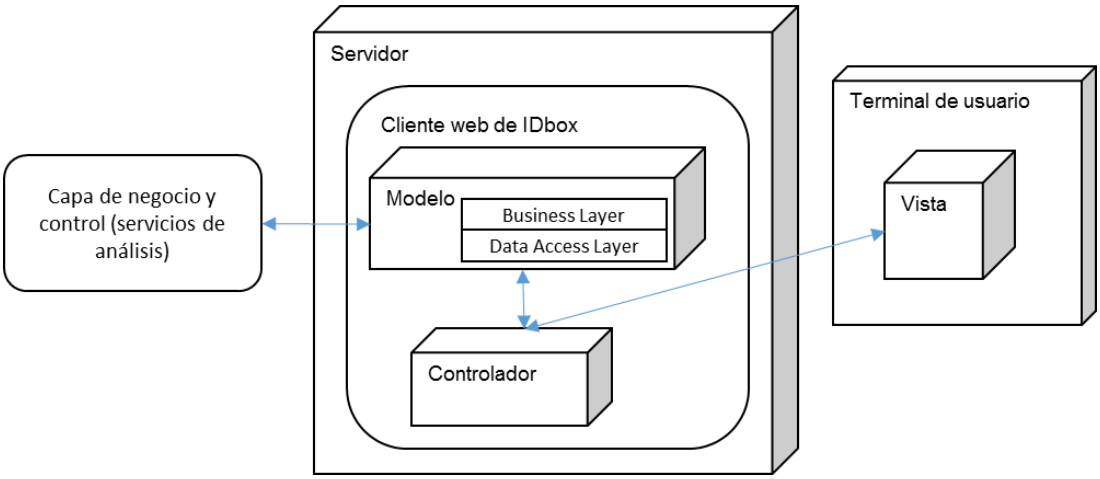

Ilustración 6: Vista de despliegue del cliente web de IDbox

Debido al uso del patrón de diseño MVC, se consigue aislar la capa de acceso a datos (Modelo) de la capa de lógica de negocio (controlador) y de la capa de visualización de datos (vista).

2.1.2 Motor de cálculo (BPM)

El motor de cálculo, representado en la parte superior de la Ilustración 4, es el encargado de procesar y tratar los datos obtenidos por los drivers según la configuración lógica dispuesta para cada uno, pudiéndolos separar a través de grupos lógicos. Los cálculos se configuran a través de una herramienta que permite, mediante una paleta de componentes, diseñar un diagrama arrastrando y soltando operadores para componer las formulas necesarias. Existen numerosos operadores para realizar cálculos: lógicos, aritméticos, cálculos matemáticos específicos, etc.

La Ilustración 7 muestra una vista general de la interfaz de usuario de la herramienta para generación de diagramas de cálculos. A continuación se describen detalladamente estos diagramas.

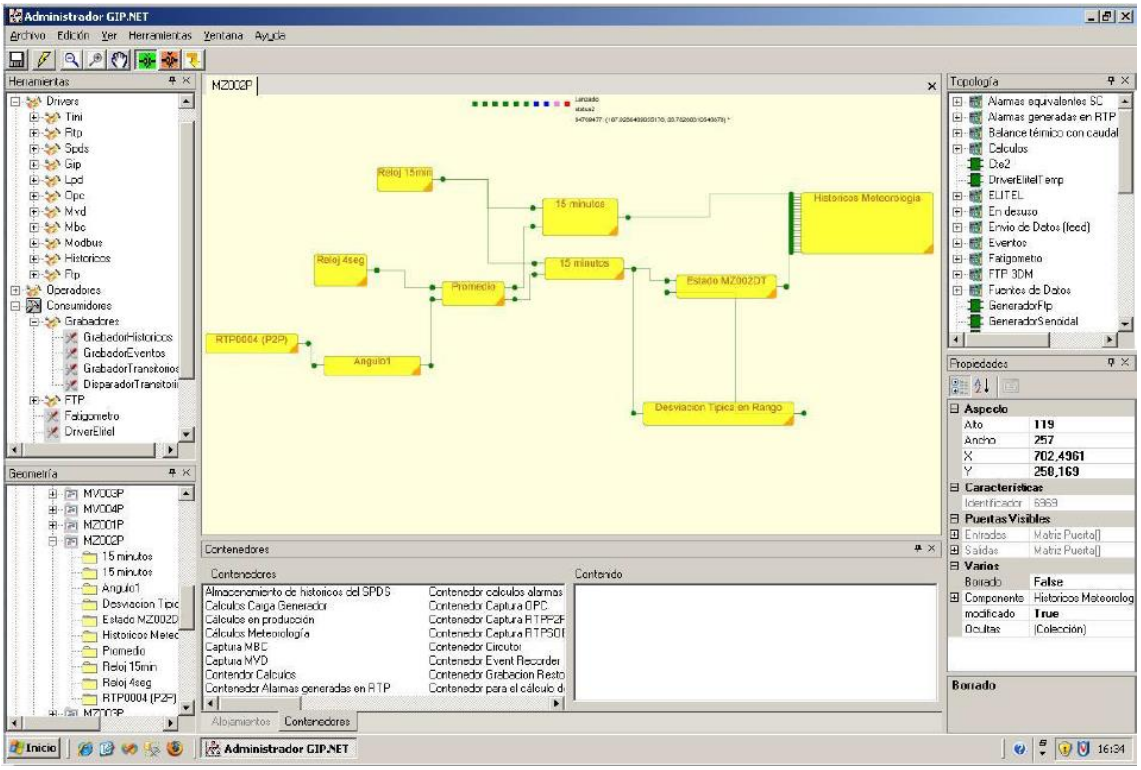

Ilustración 7: Configurador del motor de cálculo de IDbox

2.1.3 Diagramas de cálculo

Cada diagrama se corresponde con un cálculo y se representa mediante un gráfico vectorial SVG[1] [3], lo que permite adaptarse a cualquier dispositivo y equipo de visualización, realizar zoom y escalados. En la Ilustración 8, se muestra un ejemplo de un diagrama de cálculo.

[1] SVG es el acrónimo de Scalable Vector Graphics (Gráficos vectoriales escalables), un estándar para describir imágenes vectoriales bidimensionales.

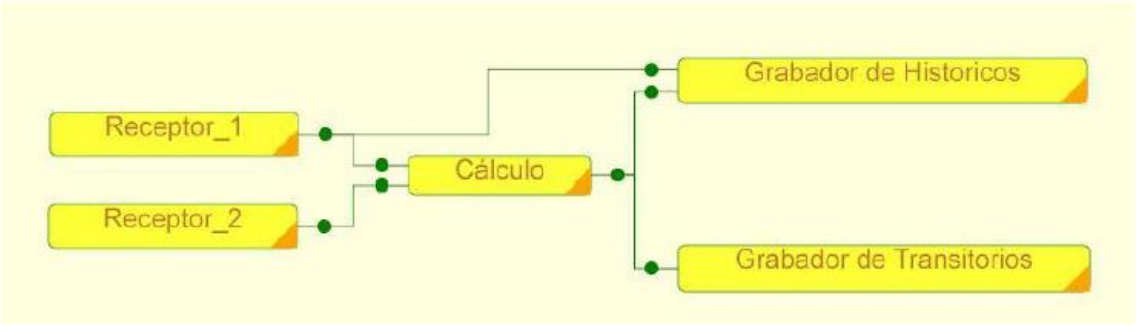

Ilustración 8: Ejemplo de diagrama de cálculo de IDbox

La representación de los cálculos está organizada como un grafo en el cual cada elemento es independiente y parametrizable. Cada elemento puede formar parte de varios cálculos y cada cálculo puede ser encapsulado y formar parte de otros.

Los elementos que forman un cálculo son los siguientes:

- Diagrama: Objeto que se corresponde con el total del cálculo.
- Componente: Objeto que se corresponde con cada uno de los rectángulos amarillos visibles en la Ilustración 8. Contiene los datos de tipo de componente.
- Puerta: Objeto que se corresponde con cada uno de los círculos verdes de la Ilustración 8. Indica los puntos de entrada y salida de datos de los componentes.
- Enlaces: Objeto que se corresponde con cada una de las líneas verdes que conectan puertas.

El acceso a la herramienta de configuración está regulado por permisos: de sólo lectura, y de lectura y escritura. Un usuario con permiso de lectura podrá visualizar un diagrama, mientras que un usuario con permiso de lectura y escritura podrá visualizarlo y editarlo.

2.1.4 Base de datos de IDbox

IDbox necesita varios componentes para guardar toda la información que recoge para su posterior procesamiento. Dicho guardado se realiza mediante los servicios de almacenamiento que se muestran en la parte inferior de la Ilustración 4. Uno de ellos, el que interviene en el configurador del motor de cálculo, es decir, dónde se almacenan los datos de las estructuras de los diagramas de cálculos, es una base de datos relacional gestionada con Microsoft SQL Server.

En este apartado se muestra una pequeña parte del modelo de base de datos de la aplicación. En la Ilustración 9 se puede observar el diagrama de base de datos creado con la herramienta Microsoft SQL Server Management Studio. Este diagrama muestra cómo almacena la aplicación los datos relacionados con los diagramas de cálculo que se representan en el configurador BPM.

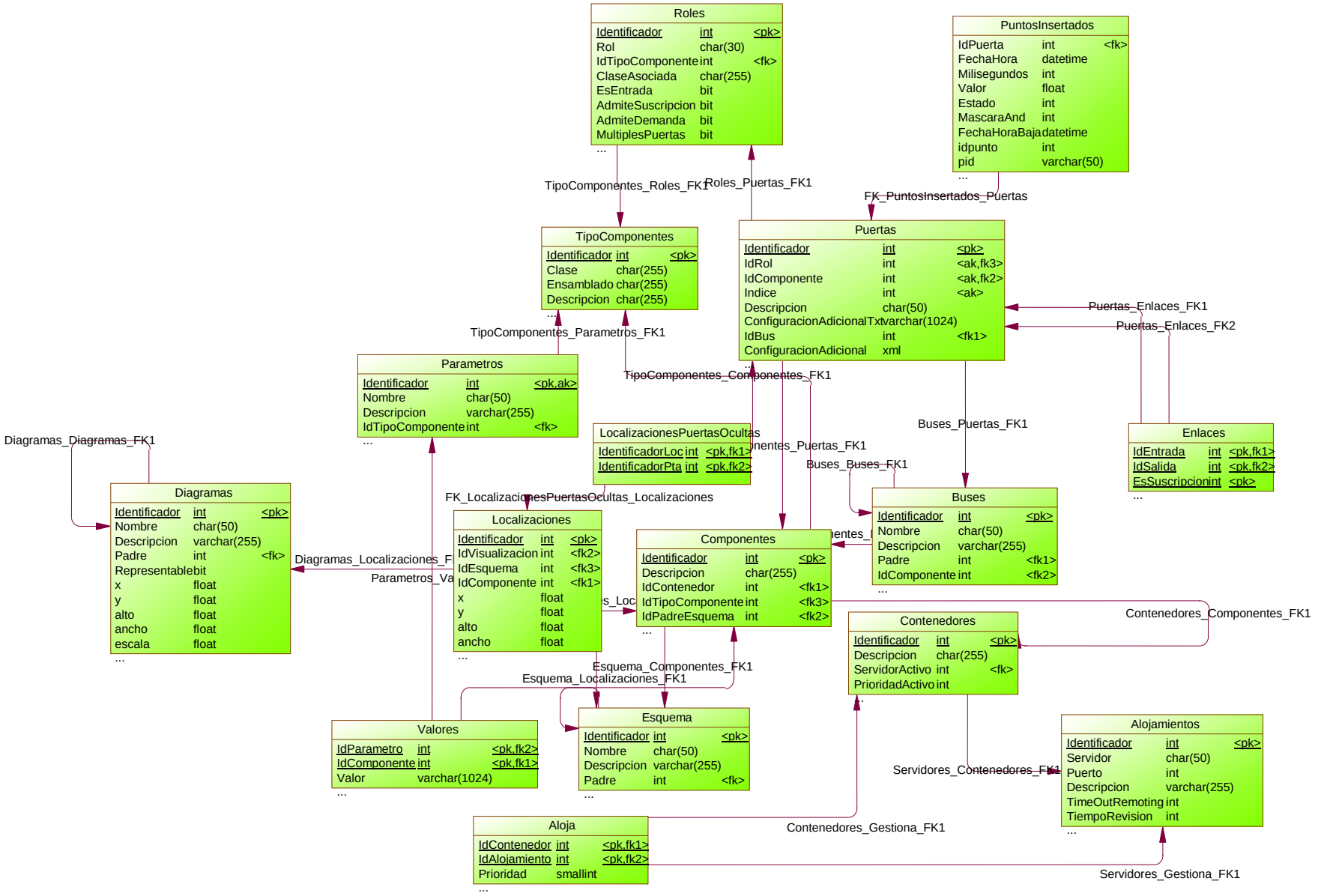

Ilustración 9: Modelo de base de datos de IDbox

Las tablas principales son:

- Diagrama: contiene los datos de visualización, como tamaño, escala de visualización, diagrama padre (si corresponde) y posición del diagrama en el lienzo de dibujo (coordenadas x e y)
- Componente: registra el tipo de componente que es y el esquema padre.
- Puerta: Indica el componente al que pertenece y el orden en el que está situado en el componente
- Enlaces: contiene la puerta de salida donde comienza un enlace, la puerta de entrada donde finaliza el enlace, y si dicho enlace es visible.
- Localizaciones: Guarda los datos relativos a un componente con cada diagrama: Posición del componente en el diagrama (coordenadas x e y) y tamaño del componente.

2.2 TECNOLOGÍAS EMPLEADAS

2.2.1 Modelo MVC

El modelo MVC [4] es un patrón de diseño ampliamente extendido entre aplicaciones que requieren de interfaces de usuario. Su objetivo es la separación del código según la responsabilidad del mismo. Dicha separación se lleva a cabo mediante la separación de sus tres componentes: el modelo, la vista y el controlador, tal y como se muestra en la Ilustración 10.

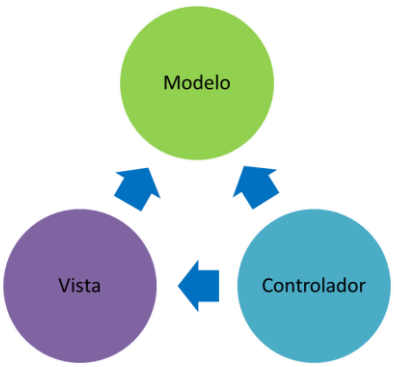

Ilustración 10: Esquema del patrón de diseño MVC

El modelo es la parte del dominio que implementa la lógica de la aplicación. Es el componente encargado de la comunicación con el sistema de almacenamiento; en este proyecto, la base de datos; en ambas direcciones, tanto para recuperar datos como para almacenarlos.

La vista es el componente que se encarga de mostrar la interfaz de usuario. Ésta se crea a partir del modelo de datos. Por ejemplo, una ventana de visualización de un diagrama de cálculo es la vista de la aplicación.

El controlador es el componente que controla la interacción del usuario, trabaja con el modelo y selecciona la vista correspondiente a mostrar en la interfaz de usuario.

El funcionamiento de una solicitud con este patrón es el siguiente: El usuario realiza peticiones que atiende el controlador, y éste último, o bien lo gestiona directamente a través de la vista (y de ahí al usuario), o bien a través del modelo y posteriormente al usuario mediante la vista. Debido a que el patrón MVC separa los tres componentes, hace posible que se paralelice el desarrollo de los mismos siempre que se hayan definido con precisión las interfaces entre ellos.

2.2.2 Arquitectura basada en capas

La arquitectura en capas (Sommerville, 2011, página 157) [5] es un patrón de diseño que, al igual que el Modelo MVC, permite obtener separación del código. Para ello, la arquitectura en capas, como su propio nombre indica, disgrega el código en n capas. Cada capa sólo se comunica con las capas adyacentes (inferior y superior) a ella, tal y como se puede observar en la Ilustración 11.

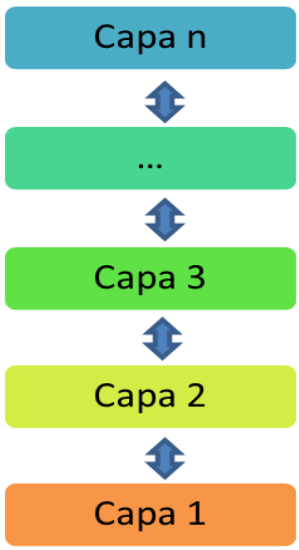

Ilustración 11: Esquema de la arquitectura basada en capas

Los desarrollos que siguen esta arquitectura son paralelizables a nivel de capa. Además, se otorga al desarrollo una alta flexibilidad de modificación, ya que se puede sustituir una capa por otra, sin obligar cambios en las capas adyacentes, siempre y cuando las interfaces de comunicación con las capas adyacentes sean iguales entre las capas sustituida y sustituta.

2.2.3 HTML5

HTML5 es una tecnología para la realización de interfaces y aplicaciones web. Habitualmente se conoce como HTML5 a la agrupación de las tecnologías HTML5+CSS3+Javascript dada la estrecha relación que hay entre ellas. Estas tres tecnologías se utilizan para tres aspectos del desarrollo claramente definidos:

- HTML: HTML es un lenguaje de etiquetado con una serie de etiquetas predefinidas y es utilizado para representar el contenido (la información) de una forma estructurada.
- CSS: CSS es un lenguaje que permite aportar estilo a la información contenida en el HTML.
- JavaScript: JavaScript es un lenguaje de programación de *scripts* (guiones) que otorga funcionalidades y dinamismo al contenido y el estilo de las aplicaciones web.

A continuación se detalla cada una de estas tecnologías.

2.2.3.1 *HTML*

HTML[2] [6] permite estructurar el contenido de una página utilizando una serie de etiquetas predefinidas. HTML permitió establecer estándares a seguir a la hora de representar información a través de protocolos HTTP mediante internet. Un ejemplo de la estructura base de HTML se muestra en la Ilustración 12:

[2]Hyper Text Markup Language es un estándar de la W3C.

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Título del documento
    </title>
  </head>
  <body>
    <p>
      Contenido de la página
    </p>
  </body>
</html>
```

Ilustración 12: Estructura básica de HTML

La estructura anterior tiene como declaración principal la etiqueta <!DOCTYPE>, la cual indica que se trata de un documento HTML. Versiones anteriores de HTML incorporaban más información referente a la versión concreta de HTML utilizada, pero en HTML5 se decidió simplificarlo dejando únicamente la etiqueta sin atributos adicionales. Dentro de esta etiqueta se realiza una división en dos secciones <head> y <body>. La sección "head" suele contener el título del documento HTML, una descripción del contenido del mismo y las dependencias con ficheros externos que pueda contener, es decir, la inclusión de ficheros CSS o JavaScript. Estos últimos también se colocan en ocasiones al final del documento HTML. Esto es debido a que al procesar el fichero HTML los ficheros con dependencias se descargan desde el servidor de forma bloqueante con lo que ante una gran cantidad de ficheros JavaScript o ficheros grandes, el procesado de la página en sí se ve retrasado hasta completar la descarga de todos los ficheros. Por otro lado, la sección "body" posee toda la información del documento HTML. Es en esta sección donde se introducen las diferentes etiquetas que permiten estructurar el contenido y que luego serán estilizadas mediante CSS. A continuación, en la Tabla 1 se detallan algunas de las etiquetas HTML utilizadas:

Etiqueta	Descripción
<table>	Permite la creación de tablas
<p>	Indica que el contenido es un párrafo de texto
	Indica una división en el contenido cuyo estilo es controlable mediante CSS
<hX>	Permite declarar cabeceras de texto (Siendo X un número entre 1 y 6)
<script>	Permite incluir código JavaScript
<img>	Permite incluir imágenes
<form>	Indica que el contenido es un formulario

Tabla 1: Algunas etiquetas HTML

2.2.3.2 *CSS*

CSS[3] [7] es un lenguaje usado para asignar estilo al contenido de documentos HTML. En su última versión, la 3, CSS presenta un formato similar al de la Ilustración 13: Ejemplo CSS

```
p {
    font-family: arial;
    font-size: 15px;
    text-align: center;
    color: navy;
    margin-bottom: 0px;
    margin-top: 40px;
}
```

Ilustración 13: Ejemplo CSS

Existen tres formas de incluir formato CSS en el código HTML:

- Utilizar un fichero CSS externo

  Esta es la opción más habitual, ya que posteriormente es posible utilizar el mismo fichero CSS para aplicar el mismo estilo a otras páginas HTML. De esta forma es más rápido realizar modificaciones en los estilos, ya que todas las páginas HTML quedarían afectadas por los cambios.

  Por un lado estaría un fichero HTML y por otro, un fichero CSS. Para enlazarlos, se debe añadir el código siguiente en el apartado <head> del fichero HTML:

  ```
  <link rel="stylesheet" href="estilo.css" type="text/css"/>
  ```

- Incluir el código CSS dentro del fichero HTML

  Este caso no precisa tener 2 ficheros, ya que todo el código está en el fichero HTML. Como desventaja frente al primer caso, en éste, el código será leído de forma secuencial. Sin embargo, la lectura del código HTML y del código CSS se realiza de forma paralela en el caso anterior.

[3]Cascading Style Sheets (Hojas de Estilo en Cascada) es un estándar de la W3C.

El código CSS se incluye en el apartado '<head>', delimitado por las dos líneas siguientes:

```
<style type="text/css">
        Micódigo CSS
</style>
```

- Aplicar CSS directamente en un texto

  Para situaciones como aplicar estilos a una frase o texto corto, puede hacerse insertando el código CSS dentro de un '' o '<p>' que encerrará el texto afectado.

  Si dicho texto tiene aplicado otro estilo a través de un fichero CSS externo o colocado en el HTML se aplicarán ambos estilos al mismo tiempo, prevaleciendo el último código ejecutado en caso de tratarse de la misma propiedad de formato sobre el mismo objeto.

  (El texto coloreado de color negro es aquel que se mostrará en pantalla con los atributos que se le hayan asignado, éstos en color azul).

```
<span style="font-size:30px;color:blue">
        Mitexto
</span>
```

2.2.3.3 *JavaScript*

JavaScript [8] (JS) es un lenguaje de programación ampliamente utilizado en el desarrollo web ya que, junto con el lenguaje HTML, dota de dinamismo e interactividad a las páginas web.

A pesar de que nació como evolución del lenguaje LiveScript, posteriormente fue estandarizado por la organización ECMA [9], creando el estándar ECMAScript [10], y JavaScript pasó a ser una implementación de este estándar.

Algunas de las características del lenguaje Javascript son:

- Interpretado: no requiere compilación. El procesamiento se realiza cuando el código correspondiente se lee.
- Débilmente tipado: Permite almacenar cualquier tipo de dato en cualquier variable.
- Orientado a objetos: Los objetos son entidades que permiten agrupar información y funciones. Esto se vuelve especialmente útil dadas las características de JavaScript que permiten almacenar funciones en variables.

En este proyecto, la tecnología JavaScript juega un papel clave, ya que será utilizada en los componentes encargados de la representación y edición de diagramas.

De una extensa gama de librerías Javascript disponibles, se ha elegido la librería JointJS [11], ya que cumple los requisitos necesarios: compatible con formato SVG, personalizable mediante su código fuente, compatible con el navegador Internet Explorer 9 y capacidad muy amplia de diseño de grafos. El funcionamiento de JointJS es sencillo. El usuario dispone de un lienzo de trabajo listo para ser personalizado, que contendrá los objetos del diagrama, en nuestro caso, componentes, puertas y enlaces entre ellas. En la Ilustración 14Ilustración 14: Ejemplo creado con Joint JS se puede ver un ejemplo de grafo creado con esta librería JS:

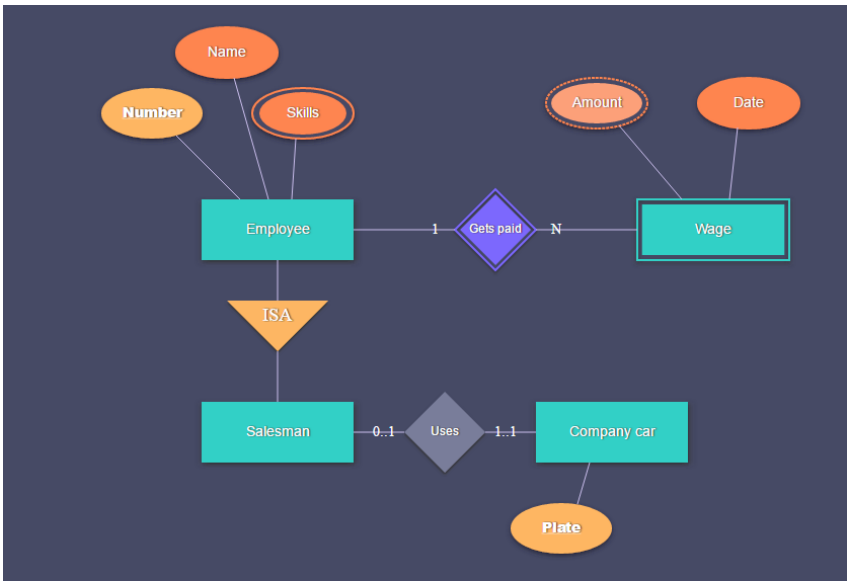

Ilustración 14: Ejemplo creado con Joint JS [12]

Para llevar a cabo un desarrollo con JointJS, está disponible una extensa API [13]. En ella, están explicadas todas las funciones de la librería, así como los modelos y variables disponibles en la misma.

2.2.4 ASP .NET MVC

ASP.NET MVC[14] es un framework basado en ASP .NET [15], que es un entorno para el desarrollo de aplicaciones web desarrollado por Microsoft. Ante la necesidad de desarrollo con la arquitectura MVC, se creó el framework ASP.NET MVC. ASP.NET MVC facilita la implementación de la arquitectura MVC sobre ASP .NET, integrando la tecnología .NET y el patrón de diseño MVC. De esta forma, este

framework ha ofrecido alternativas al desarrollo sobre .NET con las páginas ASP o los Web Forms.

Por otro lado, otro componente de ASP .NET utilizado ha sido EXT .NET [16], que se trata de un componente integrado en la librería JavaScript Sencha Ext JS. Permite la construcción de una aplicación web de visualización moderna, con arquitectura MVC y alta compatibilidad entre navegadores. Este componente mejora notablemente la interacción del usuario con la aplicación web.

3 HERRAMIENTAS Y METODOLOGÍA

Este capítulo presenta las herramientas utilizadas en el desarrollo de este proyecto, así como la metodología seguida.

3.1 HERRAMIENTAS

Las herramientas utilizadas a lo largo de este proyecto han permitido que se lleven a cabo todas las fases del mismo: análisis, desarrollo y pruebas. El uso de dichas herramientas se trata de un requisito heredado del sistema padre (requisito RH_00Captura y análisis de requisitos). Teniendo en cuenta que este proyecto es fundamentalmente un desarrollo web, las herramientas con las que se ha trabajado constan de:

- Un sistema de almacenamiento de datos: base de datos
- Un entorno de desarrollo
- Un explorador web que permita visualizar y depurar el resultado

A continuación, se detalla cada una de las herramientas:

3.1.1 Microsoft SQL Server 2012

La base de datos utilizada en el desarrollo es Microsoft SQL Server 2012 [17]. Se trata de un sistema de gestión de bases de datos relacionales. El lenguaje que utiliza es TSQL (Transact SQL).

La herramienta SQL Server Management Studio [18] permite la administración de todos los componentes de Microsoft SQL Server. En la Ilustración 15 se puede visualizar la interfaz de esta herramienta.

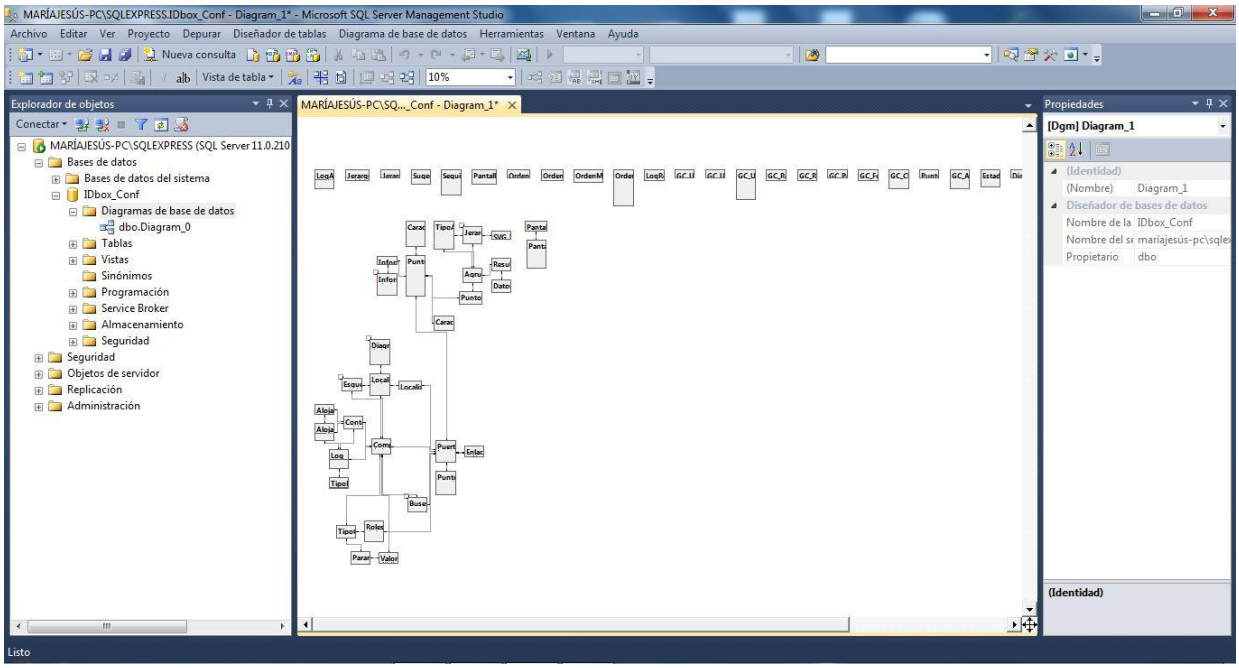

Ilustración 15: Interfaz del sistema de administración de base de datos SQL Server 2012

3.1.2 Microsoft Visual Studio 2010

Microsoft Visual Studio 2010 [19] es un entorno de desarrollo integrado (IDE, Integrated Development Environment). Esta herramienta de Microsoft permite crear múltiples tipos de aplicaciones, y junto al framework ASP .NET MVC, aplicaciones web.

Entre las características de Visual Studio, cabe destacar la variedad de lenguajes que admite (Visual Basic, C#, C++, Javascript, Python…). En el desarrollo de este proyecto se han utilizado los lenguajes C#, HTML, CSS y Javascript:

- C# es el lenguaje con el que se ha desarrollado la base del proyecto: implementación de las clases del modelo vista-controlador, así como de la arquitectura interna que rige IDbox.
- HTML es el lenguaje con el que se ha desarrollado la interfaz web.
- CSS es el lenguaje que ha aportado el estilo a la interfaz web.
- Javascript es el lenguaje utilizado en la aplicación web usando la librería JointJS.

Se puede observar una pantalla del uso de este entorno de desarrollo en la Ilustración 16.

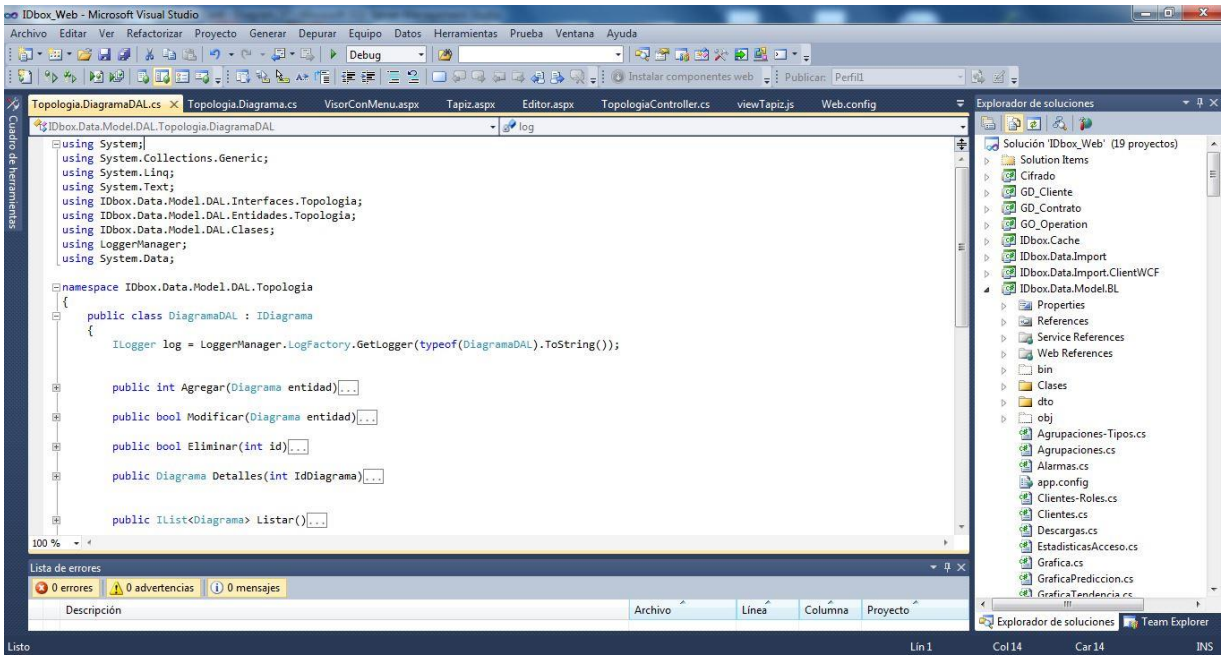

Ilustración 16: Interfaz del entorno de desarrollo Visual Studio 2010

3.1.3 Internet Explorer 9

Internet Explorer 9 [20] (IE) es un navegador de internet con el cual se visualiza el resultado del código desarrollado en el entorno de desarrollo. En concreto, sirve tanto para comprobar la visualización y funcionalidades de los objetos web, como para analizarlas, ya que dispone de una herramienta de desarrollo [21] (que se abre pulsando F12) que permite navegar por el código de la página web de forma muy intuitiva. La versión más antigua de IE sobre la que se asegura la correcta visualización del cliente web es la versión 9, por lo que este es motivo por el que se ha desarrollado haciendo uso de esta versión. En la Ilustración 17 se puede observar un ejemplo del uso de la herramienta de desarrollo de IE.

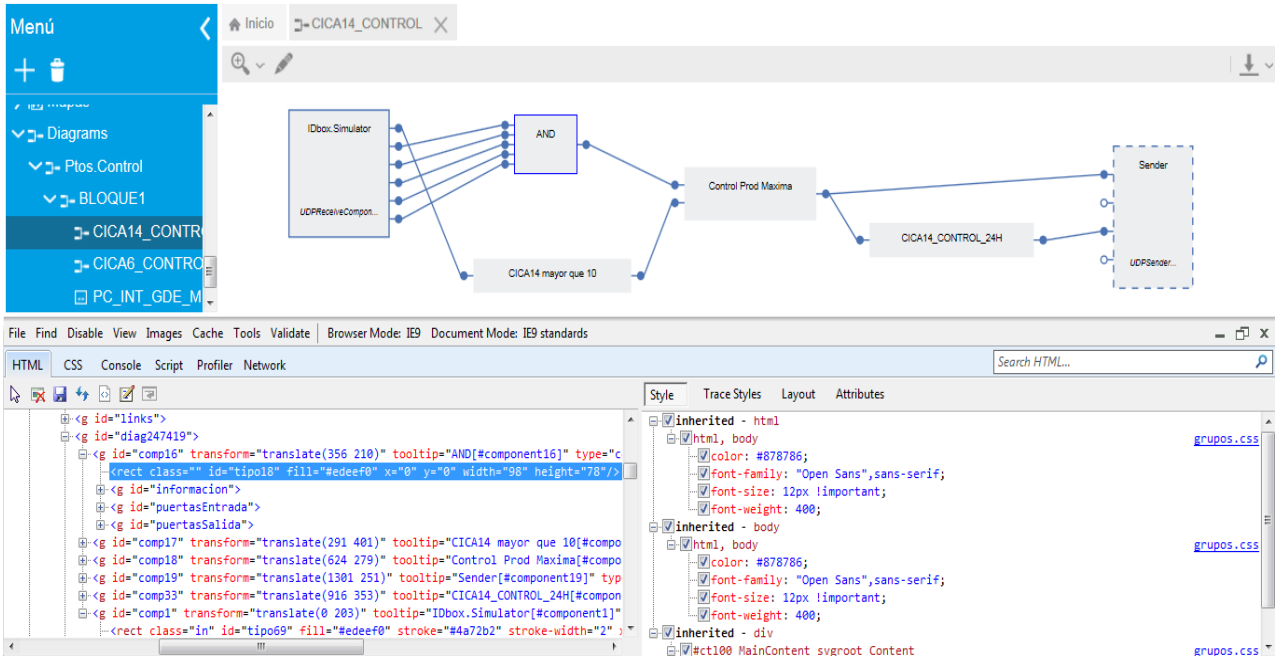

Ilustración 17: Herramienta de desarrollo del navegador IE

3.2 METODOLOGÍA DE DESARROLLO

Una decisión que marca el desarrollo es la metodología que se elige para llevar a cabo el proyecto. En este caso, la metodología de desarrollo aplicada ha sido la incremental iterativa.

> *"El desarrollo incremental se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado*."
>
> (Sommerville, 2011, página 32) [5]

El concepto principal de esta metodología consiste en la división del desarrollo total en varias partes, llamadas iteraciones o versiones. En cada iteración se establece el hito que se ha de alcanzar, siendo éste un conjunto de funcionalidades o características que se entregan al usuario al final de cada iteración para que pueda evaluarlo.

En este proyecto, tal y como se muestra en la Ilustración 18, las iteraciones desarrolladas han sido tres, tantas como grandes bloques de desarrollo ha habido en el proyecto: la primera correspondiente al tapiz, la siguiente se corresponde al visor, y la última, al editor.

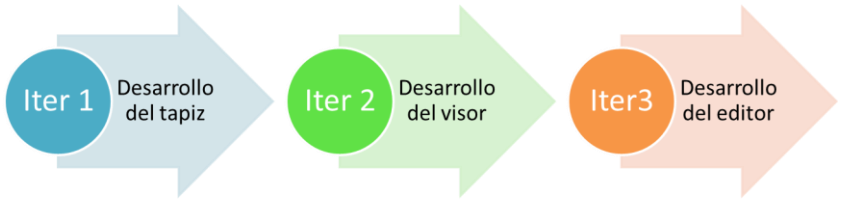

Ilustración 18: Iteraciones llevadas a cabo en el desarrollo

1. La primera iteración se corresponde con el desarrollo del componente sobre el que se dibujan los diagramas, llamado tapiz. Esta es la región sobre la que se realizan todas las posibles funciones sobre los diagramas. Las funciones que se pueden realizar en el tapiz dependen de si un diagrama se abre en modo visualización o en modo edición:
    a. Si un diagrama se abre en modo visualización, el tapiz se muestra embebido dentro del visor y las funciones disponibles sobre el diagrama del tapiz son: zoom, desplazamiento horizontal y vertical, descarga del diagrama y apertura del diagrama en el editor (ésta función sólo está disponible para los usuarios que con permisos de edición de diagramas).
    b. Si un diagrama se abre en modo edición, el tapiz se muestra embebido dentro del editor y las funciones disponibles sobre el diagrama del tapiz son: zoom, desplazamiento horizontal y vertical, descarga del diagrama, adición y supresión de componentes y enlaces al diagrama y guardado de las modificaciones.

2. La segunda iteración contempla el desarrollo del marco visor, donde se encuentra embebido el tapiz cuando se abre un diagrama en modo visualización. El menú del visor tiene las funciones de descargar diagrama y abrirlo en modo edición.

3. Finalmente, en la tercera y última iteración, se desarrolla el marco editor, donde se encuentra embebido el tapiz cuando se abre un diagrama en modo edición. El menú del editor tiene las funciones de descargar el diagrama, y las funciones de modificación: suprimir componentes y añadir nuevos. Para añadirlos, el menú del editor muestra el listado de componentes ya existentes que se pueden añadir al diagrama.

4 DESARROLLO

Este capítulo recoge el proceso de desarrollo del proyecto, incluyendo la propuesta de arquitectura del sistema, la captura y análisis de requisitos, diagramas de clase y de secuencia y muestras ejemplificativas del código resultante.

4.1 PROPUESTA DE ARQUITECTURA DENTRO DEL SISTEMA

La arquitectura del sistema está basada en los componentes que aparecen en la Ilustración 19) y permite el desarrollo de cada uno de ellos de forma independiente. En la Ilustración 19, se resalta el cliente web, ya que la integración del configurador BPM en este componente es el objetivo principal de este trabajo.

Ilustración 19: Componentes del sistema IDbox

Para integrar correctamente el nuevo desarrollo del cliente web MVC, se ha analizado la arquitectura de este componente mediante su código orientado a objetos. El patrón MVC obliga a que el código se encuentre estructurado de la siguiente forma:

- Modelo: Se compone de las funciones que recuperan los datos de base de datos y los devuelven como objetos. El modelo se compone a su vez de una arquitectura en dos capas: Business Layer (BL) y Data Access Layer (DAL). La capa DAL es la intermediaria en la comunicación entre el cliente web y el sistema de almacenamiento. Para ello, la comunicación pasa por la capa de negocio y control y la capa de servicios internos, ambas representadas en la Ilustración 19.
- Vista: está formado por el código HTML, CSS y Javascript. Además, de la librería JointJS (apartado 2.2.3.3) para la representación de grafos.
- Controlador: se corresponde con el código en C# de llamada tanto al modelo como a la vista.

4.2 CAPTURA Y ANÁLISIS DE REQUISITOS

IDbox, como herramienta ya desarrollada e implantada, cumple una serie de requisitos funcionales y no funcionales que aseguran la calidad del software. Este proyecto, al tratar el desarrollo de un prototipo integrado en la aplicación web, hereda una serie de requisitos de la aplicación padre. A continuación, se muestran los requisitos heredados (RH) del sistema padre en la Tabla 2:

Referencia	Descripción
RH_00	El desarrollo se realizará con el conjunto de herramientas ya utilizadas en el desarrollo del resto de la aplicación: Microsoft SQL server 2012 y Visual Studio 2010
RH_01	El desarrollo se realizará siguiendo la arquitectura MVC, con la misma estructura que el resto de la aplicación.
RH_02	La interfaz desarrollada deberá seguir el mismo tema de colores y tipografía que el resto de la aplicación.
RH_03	El desarrollo incluirá acceso por permisos. Un usuario de lectura podrá visualizar los diagramas, y un usuario de lectura y escritura podrá visualizarlos y editarlos.
RH_04	La aplicación desarrollada será accesible desde el navegador web Internet Explorer. La versión más antigua de IE desde la que se podrá acceder a la aplicación será la versión 9.

Tabla 2: Requisitos heredados del sistema padre

A continuación, se muestran los requisitos funcionales y no funcionales específicos para este proyecto. Por un lado, los requisitos funcionales (RF) expresan la naturaleza del funcionamiento del sistema. Detallan cómo el sistema interacciona con su entorno y cuál va a ser su estado y su funcionamiento. En la Tabla 3 se expone la lista con los requisitos funcionales de este proyecto:

Referencia	Descripción
RF_00	El sistema debe permitir la visualización de los diagramas de cálculo almacenados en base de datos.
RF_01	En cada diagrama, el sistema debe mostrar el nombre de los componentes que lo conforman.
RF_02	El sistema debe mostrar las puertas que pertenecen a cada componente en un diagrama concreto.
RF_03	Cada puerta debe tener la opción de que el sistema muestre un mensaje indicando el tipo de componente, rol, identificador y valor en tiempo real.
RF_04	El sistema debe ofrecer la posibilidad de exportar un diagrama en formato SVG.
RF_05	El sistema debe permitir la edición de los diagramas almacenados en base de datos.
RF_06	El sistema debe permitir realizar zoom en la visualización de un diagrama
RF_07	El sistema debe permitir hacer scroll para desplazarse por el diagrama (cuando el diagrama sea más grande que el área de visualización)
RF_08	El sistema debe contar con un área de vista previa que muestre el diagrama a escala en la pantalla de edición (dicha escala se calculará según el tamaño del área de vista previa, que dependerá del tamaño de la pantalla de visualización).

Tabla 3: Requisitos funcionales específicos del sistema

Por otro lado, los requisitos no funcionales (RNF) desempeñan un papel fundamental en el desarrollo de productos de software. No se refieren a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste, como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento [5]. Se ha utilizado el estándar ISO 25010 [24] para realizar la clasificación de los requisitos.

Referencia	Descripción	Importancia
Rendimiento		
RNF_01	El sistema debe contemplar la conexión concurrente de cientos de usuarios	Alta
Usabilidad		
RNF_02	Tras una navegación inicial de 15 minutos, un usuario tiene que ser capaz de utilizar todas las funciones del sistema.	Alta
RNF_03	La interfaz debe ser intuitiva para el usuario.	Alta
Disponibilidad		
RNF_04	El sistema debe estar disponible el 95% del tiempo, siendo el 5 % restante para tareas de mantenimiento.	Alta
Seguridad		
RNF_05	Los usuarios podrán acceder a los módulos y funciones en función de sus permisos.	Muy alta

Tabla 4: Requisitos no funcionales específicos del sistema

4.3 DIAGRAMAS DE SECUENCIA

Los diagramas de secuencia son usados para representar las interacciones entre los actores y objetos que intervienen en un determinado proceso.

En este desarrollo, se han elegido dos interacciones concretas de un usuario en el cliente web para representarlas como diagramas de secuencia:

- La primera, consiste en la visualización de un diagrama elegido por un usuario. Para ello, un usuario que ha accedido al sistema, visualiza el listado de diagramas existentes, y elige abrir uno de ellos. Posteriormente, se dibuja el diagrama seleccionado en el tapiz.

  Los pasos serían los siguientes:
  1. Un usuario, logado en la web, despliega el listado de diagramas.
  2. Se carga el listado de diagramas.
  3. El usuario elige abrir un diagrama de entre los posibles.
  4. Se carga el diagrama seleccionado en el tapiz del visor.

En la Ilustración 20 se puede consultar el diagrama de secuencia asociado.

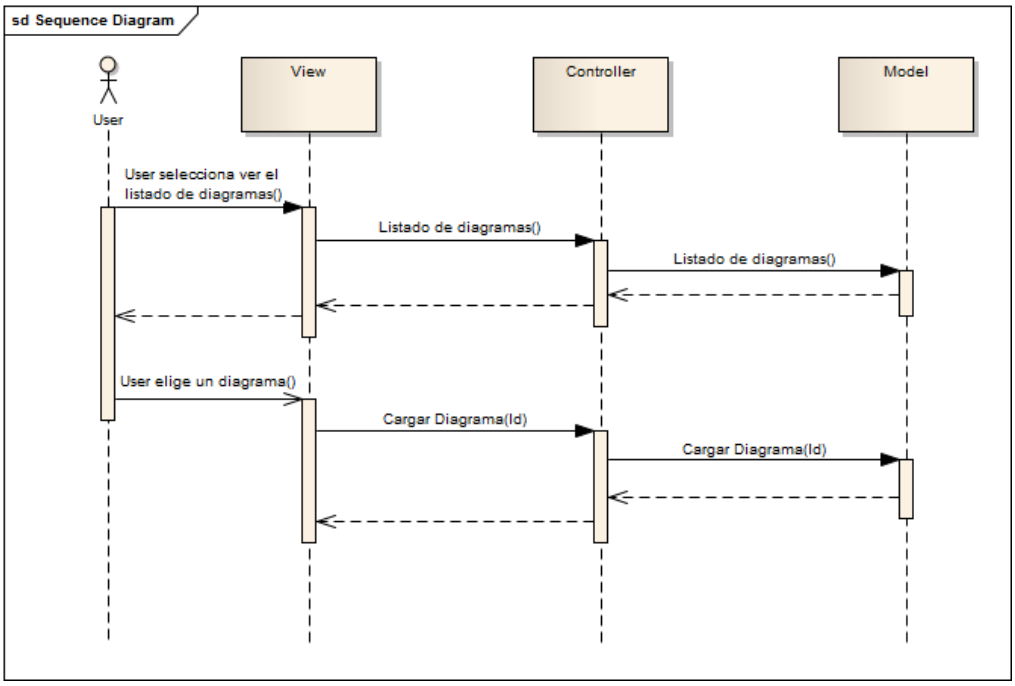

Ilustración 20: Diagrama de secuencia de visualización de un diagrama

- La segunda interacción consiste en la consulta del valor en tiempo real de una puerta elegid por el usuario dentro del diagrama que se está visualizando. Para ello, un usuario que ha accedido a la visualización de un diagrama, elige abrir el menú de detalle de una puerta. Posteriormente, en un plazo máximo de 3 segundos, aparece el valor en tiempo real (TR) de la puerta en cuestión, que se refresca automáticamente cada 3 segundos.

  Los pasos serían los siguientes:
  1. Un usuario está visualizando un diagrama
  2. El usuario elige una puerta del diagrama y abre el menú de detalle de esta puerta.
  3. El valor en tiempo real de la puerta aparecerá a los 3 segundos o menos (debido a que el listado de valores en tiempo real de las puertas se actualiza con una frecuencia de 3 segundos).

  En la Ilustración 21 se puede visualizar el diagrama de secuencia correspondiente a esta interacción.

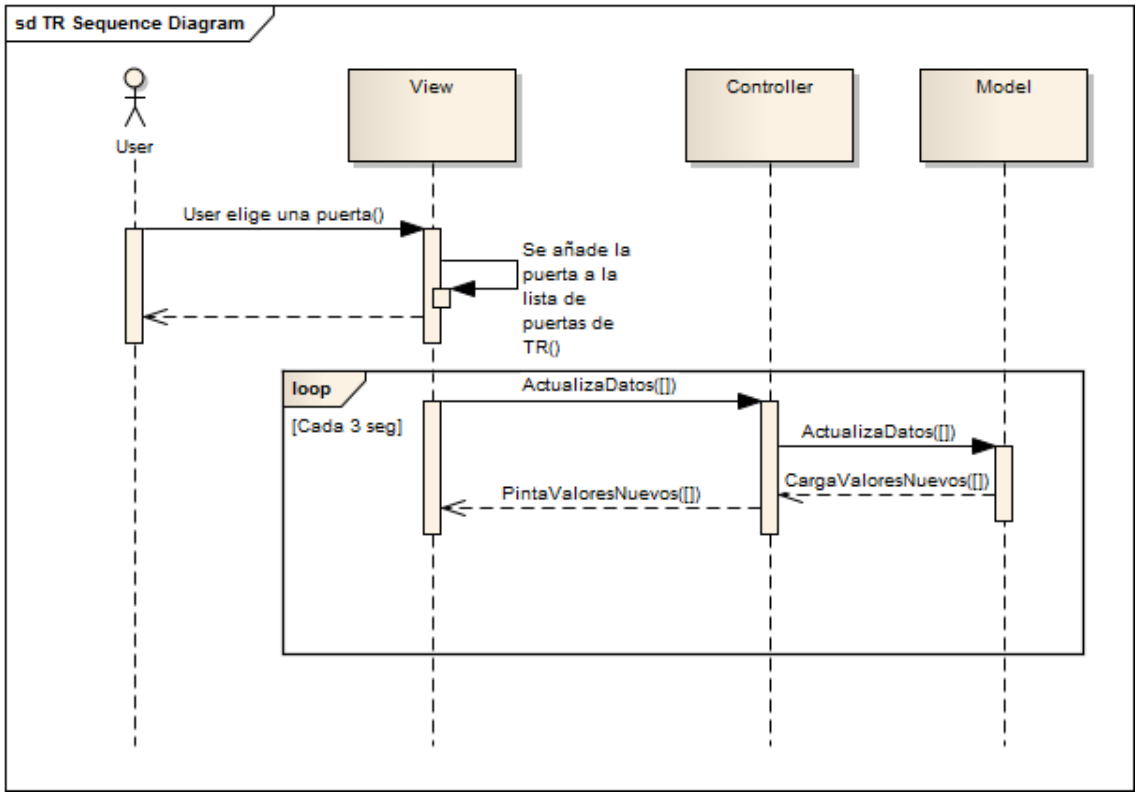

Ilustración 21: Diagrama de secuencia de consulta del valor TR de una puerta

4.4 DIAGRAMA DE CLASES

Los modelos estructurales de software muestran la organización de un sistema, concretamente, los componentes que constituyen dicho sistema y sus relaciones. Los modelos estructurales pueden ser modelos estáticos, que muestran la estructura del diseño del sistema, o modelos dinámicos, que revelan la organización del sistema cuando se ejecuta [5]. Un diagrama de clases UML[4] es un modelo estructural estático que muestra las clases de un sistema y las relaciones entre ellas, como herencia, asociación, agregación, etc. Los diagramas de clases pueden usarse cuando se desarrolla un modelo de sistema orientado a objetos para mostrar las clases en un sistema y las asociaciones entre dichas clases. A grandes rasgos, una clase de objeto se considera como una definición general de un tipo de objeto del sistema. Una asociación es un vínculo entre clases, que indica que hay una relación entre las

[4] Unified Modeling Language (Lenguaje unificado de modelado) [25] es un lenguaje de modelado software estandarizado por el OMG (Object Management Group) [26]

mismas. En la Ilustración 22 se puede consultar el diagrama de clases creado para representar la estructura del sistema. Se puede ver cómo el diagrama se separa en 3 regiones: modelo, controlador y vista. A su vez, se distingue fácilmente la arquitectura en dos capas dentro del modelo: por un lado la capa de acceso a datos (DAL) y por otro lado, la capa de negocio (BL). La capa DAL se estructura de la siguiente forma:

- En la primera línea del diagrama se encuéntrala clase IDAL, que es la interfaz padre que hederan todas las interfaces hijas.
- En la segunda línea del diagrama se encuentran las interfaces que heredan de la interfaz IDAL. Existe una interfaz por cada tipo de objeto (diagrama, componente, conexión, puerta y propiedades). Cada interfaz contiene la estructura de los métodos que ha de implementar la clase hija.
- En la tercera línea del diagrama se encuentran las clases DiagramaDAL, ComponenteDAL, etc. Cada una de ellas se implementa según su interfaz correspondiente.

La capa BL devuelve al controlador los objetos para que puedan ser representados en la vista. Para facilitar la edición de los atributos de las clases, la capa BL se divide en dos secciones: por un lado las clases DTO, que contienen los atributos y los respectivos métodos get y set; y por otro lado las clases que contienen los métodos que implementan las funcionalidades requeridas para cada una.

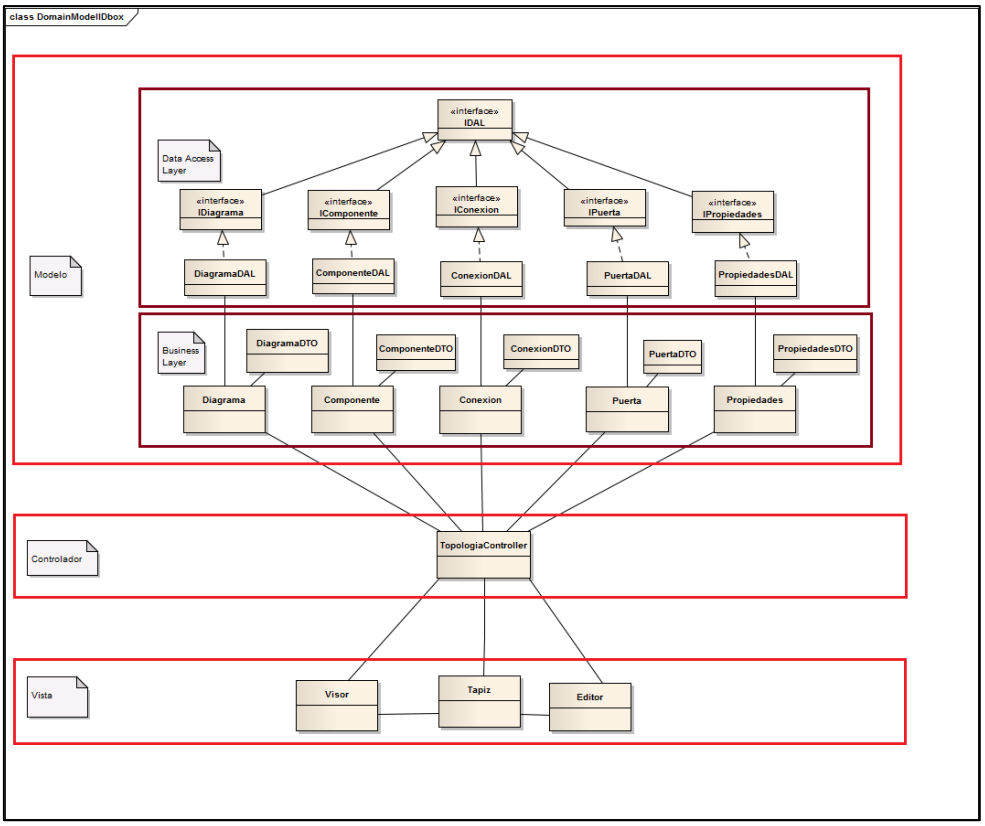

Ilustración 22: Diagrama de clases del sistema

4.5 CÓDIGO Y PRUEBAS

En este apartado se muestra un ejemplo del código desarrollado así como el resultado obtenido (interfaz) para las dos mismas interacciones explicadas en el capítulo 4.3 Diagramas de secuencia: Visualización de un diagrama y consulta del valor en tiempo real de una puerta. El código desarrollado se desglosa según los patrones de diseño utilizados: modelo (dos capas), controlador y vista.

4.5.1 Interacción 1: Visualización de un diagrama

Esta interacción consiste en la visualización de un diagrama, previamente guardado en base de datos.

4.5.1.1 *Modelo*

El modelo contiene el código que llama a la base de datos.

Capa de acceso a datos (Data Access Layer):

La capa de acceso a datos recupera los datos de base de datos.

```
/* Este método devuelve un diagrama después de recibir como
 param un ID de diagrama
 PARAMETROS: int
 RETURN: Diagrama
 */

public Diagrama Detalles(int IdDiagrama)
{
        log.Information("DAL.Topologia.cs - Acceso al método
        Diagrama_Datos({0})", IdDiagrama);
        Diagrama diag = new Diagrama();
        try
        {
                log.Debug("Creando consulta");
                string sql = string.Format(
                   " select  " +
                   "     nombre, " +
                   "     ancho, " +
                   "     alto, " +
                   "     escala, " +
                   "     x, " +
                   "     y, " +
                   "     representable " +
                   " from diagramas " +
                   " where identificador = {0}", IdDiagrama);
                log.Debug("Creando conexión a la base de datos");
                AccesoDatos bd = new AccesoDatos();
                log.Debug("Ejecutando consulta");
                DataSet ds = bd.FillDataSet(sql, false);
                if ((ds != null) && (ds.Tables.Count > 0))
                {
                        log.Debug("Datos obtenidos");
                        diag = ParsearFila(ds.Tables[0].Rows[0]);
                }
                else
                {
                        log.Warning("No se ha obtenido ningún dato");
```

```
                }
        }
        catch (Exception er)
        {
                log.Error("DAL.Diagrama_Datos()", er);
        }
        log.Information("DAL.Topologia.cs - Método Diagrama_Datos() terminado.
        Saliendo de él.");
        diag.Id_Diagrama = IdDiagrama;
        return diag;
}
```

Capa de negocio (Business Layer):

La capa de negocio llama a la capa de acceso a datos y devuelve los datos al controlador.

```
/* Este método devuelve un DiagramaDTO después de recibir como
 param un ID de diagrama
 PARAMETROS: int
 RETURN: DiagramaDTO
 */

public DiagramaDTO Detalles(int Id_Diagrama)
{
        DiagramaDTO diag = null;

        try
        {
                DAL.Topologia.DiagramaDAL diagramaDAL =
                new DAL.Topologia.DiagramaDAL();
                DAL.Entidades.Topologia.Diagrama entDiag =
                diagramaDAL.Detalles(Id_Diagrama);
                diag = ConvertirDiagramaDAL(entDiag);
                DAL.Topologia.ComponenteDAL componentesDAL =
                new DAL.Topologia.ComponenteDAL();
                DAL.Topologia.ConexionDAL conexionesDAL =
                new DAL.Topologia.ConexionDAL();
                List<DAL.Entidades.Topologia.Componente> lstCmpDAL =
                new List<DAL.Entidades.Topologia.Componente>();
                List<DAL.Entidades.Topologia.Conexion> lstCnxDAL =
                new List<DAL.Entidades.Topologia.Conexion>();
                lstCmpDAL = componentesDAL.Listar(Id_Diagrama);
                lstCnxDAL = conexionesDAL.Listar(Id_Diagrama);
                List<ComponenteDTO> lstComponentesDTO =
                Diagrama.ConvertirComponenteDAL(lstCmpDAL);
                List<ConexionDTO> lstConexionesDTO =
                Conexion.ConvertirConexionDAL(lstCnxDAL);
                diag.componentes = lstComponentesDTO;
                diag.conexiones = lstConexionesDTO;
        }
        catch (Exception er)
        {
                log.Error("Detalles()", er);
        }
        return diag;
}
```

4.5.1.2 *Controlador*

El controlador hace la llamada al modelo y devuelve a la vista el resultado.

```
/* Este método devuelve a la vista un diagrama después de recibir como
 param un ID de diagrama
```

```
 PARAMETROS: int
 RETURN: ActionResult
 */


public ActionResult CargarDiagrama(int Id_Diagrama)
{
        log.Information("TopologiaController.cs - Acceso al método CargarDiagrama({0})",
        Id_Diagrama);
        AjaxStoreResult r = new AjaxStoreResult();
        try
        {
                IDbox.Data.Model.BL.Topologia.Diagrama topologia = new
                IDbox.Data.Model.BL.Topologia.Diagrama();
                log.Debug("Llamada IDbox.Data.Model.BL.
                Topologia.Diagrama.Detalles({0}")
                Obteniendo el diagrama", Id_Diagrama);
                var diagrama = topologia.Detalles(Id_Diagrama);
                log.Debug("Retornando el diagrama {0}",diagrama);
                r = new AjaxStoreResult(new { Diagrama = diagrama });
        }
        catch (Exception er)
        {
                log.Error("TopologiaController.cs:
                CargarDiagrama({0})",
                er, Id_Diagrama);
        }
        log.Information("TopologiaController.cs - Método CargarDiagrama({0})
        terminado. Saliendo de él.", Id_Diagrama);
        return r;
}
```

4.5.1.3 *Vista*

El código JS de la vista es quien se encarga de hacer la llamada al controlador para que recupere el diagrama de base de datos, y posteriormente dibuja el diagrama.

```
/* Esta función establece el modo de visualización
o de edición en el tapiz y llama a la función CargarDiagrama.
 param un ID de diagrama
 PARAMETROS:
 RETURN:
 */

function Carga_todo()
{
        InicializaTapiz();
        var editor= window.parent.Ext.getCmp('pnlPanelDibujo');
        if(editor!= undefined)
                SetModoVisualizacionFalse();
        else{
                var visor= window.parent.Ext.getCmp('pnaSVGConenendor');
                if ( visor != undefined)
                        SetModoVisualizacionTrue();
        }
        CargarDiagrama(hdnID.getValue());
}


/* Esta función pinta un diagrama en el tapiz
cuando recibe un IdDiagrama como parámetro
 PARAMETROS: IdDiagrama
```

```
 RETURN:
 */
function CargarDiagrama(idDiagrama) {
	try
	{
		if( idDiagrama != null || idDiagrama != undefined)
		{
			MostrarCargando("Cargando", true);

			Ext.Ajax.request({
				url: '/Topologia/CargarDiagrama',
				params: {
					Id_Diagrama: idDiagrama
				},
				success: function (response) {
				try {
					var diagrama =
					Ext.decode(response.responseText).data;
					LimpiarTapiz();
					InicializaLienzo(diagrama.Diagrama);
					PintarDiagrama(diagrama.Diagrama);
					EstablecerEscala(diagrama.Diagrama.Escala);
					Inicializa_VisorSinopticos();
					MostrarCargando("", false);
					EstablecerEscala(diagrama.Diagrama.Escala);
					CentrarEnPantalla(true, diagrama);
				}
				catch (e) {
					MostrarCargando("", false);
					alert('Tapiz: No se ha podido cargar el
					diagrama correctamente. '
					+ e.message);
				}
				},
				failure: function (response) {
					MostrarCargando("", false);
					alert('Tapiz: Error en la carga de datos.'
					+ response.Message);
				}
			});
		} // if
	} //try
	catch (e) {
		MostrarCargando("", false);
	}
}
```

4.5.1.4 *Resultado: Interfaz*

Tras la selección de un diagrama, y de la ejecución del código anterior, en la Ilustración 23 se puede ver que el diagrama elegido se muestra en el tapiz del visor.

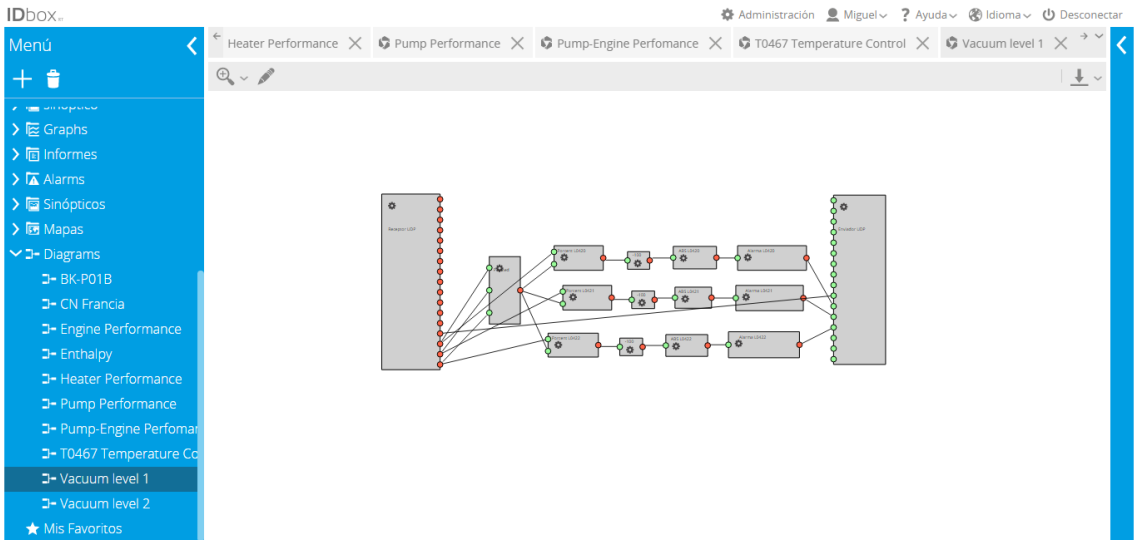

Ilustración 23: Interacción 1: visualización de diagrama

Además si se despliega el panel derecho, se puede ver el área de vista previa, en el que el diagrama se encuentra dibujado a escala, tal y como se aprecia en la Ilustración 24.

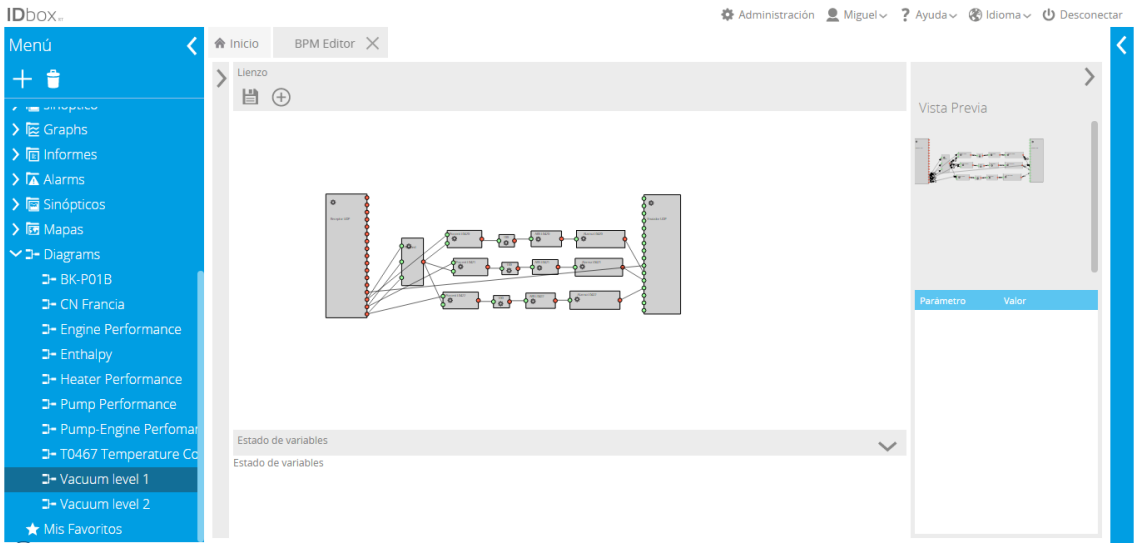

Ilustración 24: Interacción 1: visualización de diagrama con vista previa

4.5.2 Interacción 2: Consulta del valor en tiempo real (TR) de una puerta

Esta interacción consiste en la consulta del valor en tiempo real de una puerta.

4.5.2.1 *Modelo*

En este caso, los datos no se consultan directamente a base de datos, sino que se hace a través de un servicio web (Web Service) [27]. Por tanto, la cadena de llamadas llega hasta la capa de negocio (BL), pero no a la capa de acceso a datos (DAL).

Capa de negocio (Business Layer)

```
/* Este método recibe un array de puertas
para las cuales devuelve su correspondiente
valor en tiempo real, en la misma posición
del array en la que se recibe cada puerta.
 PARAMETROS: int[]
 RETURN: double[]
 */

 public double[] DatosTR(int[] puertas)
{
        WS_GestionTR.GestionTR ws = new WS_GestionTR.GestionTR();
        double[] valores = new double[puertas.Length];
        for (int i = 0; i < puertas.Length; i++)
        {
                        int id = puertas[i];
                        var r = ws.EstadoPuerta(id);
                        valores[i]= (Object) r;
        }
        return valores;
}
```

4.5.2.2 *Controlador*

```
/* Este método del controlador recibe un array de puertas
para las cuales consulta su valor en tiempo real
y lo devuelve a la vista como ActionResult
 PARAMETROS: int[]
 RETURN: ActionResult
 */

public ActionResult DatosTR( int [] puertas)
{
        log.Information("TopologiaController.cs - Acceso
        al método DatosTR()");
        AjaxStoreResult r = new AjaxStoreResult();
        try
        {
                if (puertas.Length > 0) // pido datos si hay algun tooltip abierto
                {
                        if (log.isTraceEnabled)
                        {
                                log.Trace("Listado de puertas:");
                                foreach (int puerta in puertas)
                                {
                                }
                                        log.Trace(puerta.ToString());
                        }
```

```
                        log.Debug("La lista de puertas no está vacía,
                        llamo a Models.Topologia.DatosTR() y obtengo los datos");
                        IDbox.Data.Model.BL.Topologia.Diagrama topologia = new
                        IDbox.Data.Model.BL.Topologia.Diagrama();
                        var datos = topologia.DatosTR(puertas);
                        log.Debug("Retornando los datos obtenidos");
                        if (log.isTraceEnabled)
                        {
                                log.Trace("Datos:");
                                foreach (Object dato in datos)
                                {
                                        log.Trace(dato.ToString());
                                }
                        }
                        r = new AjaxStoreResult(datos);
                }
                else
                {
                        log.Warning("No se ha pasado ninguna puerta");
                }
        }
        catch (Exception er)
        {
                log.Error("TopologiaController.cs: DatosTR()", er);
        }
        log.Information("TopologiaController.cs - Método DatosTR()
        terminado. Saliendo de él.");
        return r;
}
```

4.5.2.3 *Vista*

El código JS de la vista se encarga de hacer la llamada al controlador

```
/* Esta función agrega una puerta
al array diccionario_TR de puertas de tiempo real
cuando se hace click sobre ella
 PARAMETROS: int
 RETURN:
 */

function AgregarPuertaTR(idPuerta)
{
 try{
                var puerta= getPuerta(idPuerta);
                if(puerta.visible==1){
                        diccionario_TR.push(idPuerta);
                        var texto = puerta.Descripcion.trim() + '\nRol: '
                        + puerta.rol.trim() +
                        '\nId: ' + puerta.Identificador + '\nValor: [#'
                        + puerta.Identificador +']';
                        var titulo = '';
                        var puerta =
                        {
                                Id_Puerta: idPuerta,
                                TextoBase : texto,
                                Tooltip: texto,
                                text: titulo
                        };
                        puerta.magnet=true;
```

```
                        _Listado_Puertas.push(puerta);
                        var correspondenciaComponentePuerta =
                        {
                                Componente_Id: componente.Id_Componente,
                                Puerta_Id: puerta.Identificador
                        }
                        _diccionario_Componente_Puerta.push(correspondenciaComponentePuerta);
                } // if puerta visible == 1
        } //try
        catch(e){
                Log.error('AgregarPuertaTR(). ' + e.message);
                alert(e);
        }
}


/* Esta función hace la llamada al controlador
cuando se hace clic sobre una puerta
Envía como parámetro el array diccionario_TR
 PARAMETROS:
 RETURN:
 */


function Datos_TR()
{
        try
        {
                Ext.Ajax.request({
                        url: '/Topologia/DatosTR',
                        params: {
                                        puertas: diccionario_TR
                                },
                        success: function (response) {
                                try {
                                        var valores = Ext.decode(response.responseText).data;
                                        ActualizaTooltips(valores);
                                }
                                catch (e) {                     alert('ViewTapiz.Datos_TR: No se
                        ha podido cargar el valor TR de la puerta.'
                                        + e.message);
                                }
                        },
                        failure: function (response) {
                                alert('ViewTapiz.Datos_TR: No se ha podido
                                cargar el valor TR de la puerta.'
                                + response.Message);
                        }
                });
        }
        catch (e) {}
}


/* Esta función actualiza el valor de TR
de las puertas que estan en el array diccionario_TR
Recibe como parametro el array de valores TR.
Para calcular los valores, se mapean las posiciones en los arrays:
el valor correspondiente a diccionario_TR[i] es valores_nuevos[i]
 PARAMETROS: double[]
 RETURN:
 */


function ActualizaTooltips (valores_nuevos)
{
```

```
        //recorro diccionario_TR y valores nuevos y llamo a  ActualizaTooltip (idPuerta, valor)
con cada pareja
        for (var i = 0  ; i < diccionario_TR.length ; i++)
        {
                idPuerta=diccionario_TR[i];
                valor=valores_nuevos[i];
                ActualizaTooltip (idPuerta, valor);
        }
}


/* Esta función actualiza el texto del tooltip
de cada puerta en diccionario_TR
 PARAMETROS: int, double
 RETURN:
 */


function ActualizaTooltip (idPuerta, valor)
{
        var puerta = getPuerta_byidbd(idPuerta) ;
        var textoBase = puerta.TextoBase
        if (valor == undefined)
                valor = "***";
        puerta.Tooltip = textoBase + valor;
        var a = 0;
}
```

4.5.2.4 *Resultado: Interfaz*

Tras hacer clic sobre una puerta, ésta se añade al listado de puertas de las que se cargan su valores en tiempo real. Posteriormente, aparece un cuadro emergente (tooltip) al lado de la puerta con la información de ésta y el valor en tiempo real. En la Ilustración 25 se puede apreciar un diagrama con valores de dos puertas.

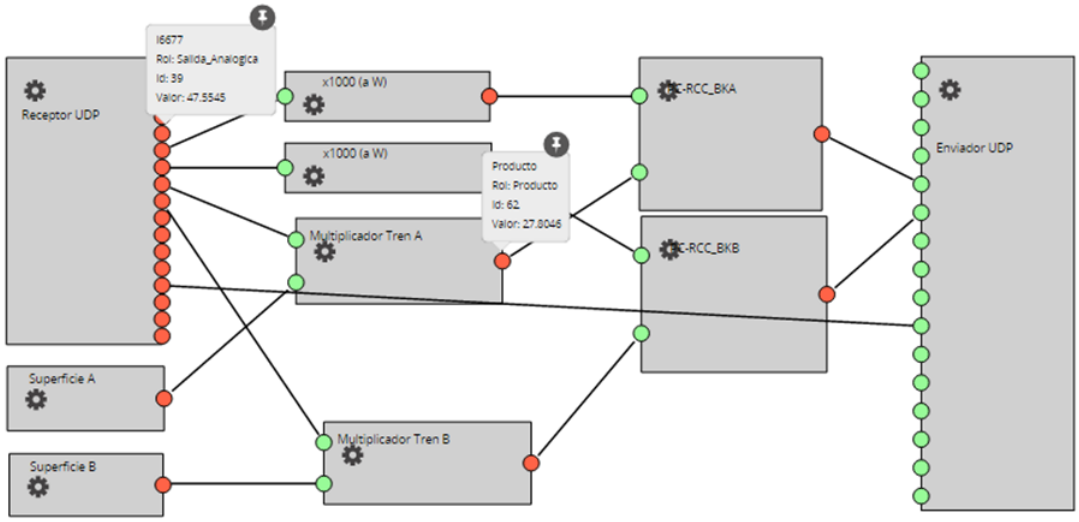

Ilustración 25: Interacción 2: visualización de valores en TR de puertas

5 CONCLUSIONES

El objetivo de este proyecto fin de carrera ha sido la implementación de un diseño que integra el componente BPM Engine de IDbox en un cliente web. Este objetivo inicial resultó ambicioso dado el tiempo disponible, por lo que no ha sido cumplido en su totalidad. El desarrollo de los tres componentes (tapiz, visor y editor) en los que se divide este proyecto se completó satisfactoriamente. Asimismo, se implementaron las funcionalidades de visualización y edición de diagramas para el motor de cálculo de IDbox. Sin embargo, la funcionalidad de creación de diagramas desde cero en el editor no se llegó a implementar y se ha dejado como trabajo futuro.

. Al término de este proyecto, se han obtenido una serie de conclusiones:

- En cuanto a las necesidades de las empresas, cabe destacar la importancia de adaptar los sistemas al tipo de tecnología de hoy en día. Queda clara la necesidad latente que tienen las empresas de que antiguas aplicaciones de escritorio se conviertan en sistemas flexibles y accesibles vía web.

- En cuanto a la metodología seguida, cabe destacar la importancia de la elección de la metodología que más se ajusta a las necesidades del desarrollo y del cliente.

  En este caso, llevar a cabo el proyecto con metodología iterativa ha otorgado flexibilidad al desarrollo, ya que ha permitido detectar nuevos requisitos no contemplados anteriormente, y obtener feedback del cliente: mejoras de la iteración anterior.

- En cuanto a las tecnologías utilizadas, cabe reseñar que el entorno de desarrollo Visual Studio ofrece una gran variedad de funcionalidades que facilitan sustancialmente la tarea de programación.

- Finalmente, se ha conseguido crear una aplicación web capaz de realizar de forma parcial las funciones de una aplicación de escritorio diseñada para el sistema SCADA IDbox. Esta aplicación web es muy útil en cuanto la movilidad que otorga a los usuarios, así como la rapidez de respuesta ante una alarma, ya que no se precisa que el usuario acuda al lugar geográfico donde haya un dispositivo con la aplicación de escritorio instalada. De esta forma, el sistema IDbox ofrece una funcionalidad más, que lo diferencia de un sistema SCADA tradicional.

6 LÍNEAS FUTURAS

A medida que se llevaba a cabo el desarrollo del presente proyecto, se detectaban una serie de posibles mejoras. A continuación, se enumeran dichas mejoras:

- Creación de un diagrama desde cero.

  Este proyecto ha abarcado tanto la visualización de diagramas, como la edición de los mismos. Por ello, una línea de futuro es el desarrollo que permita la creación de diagramas desde cero.

- Modificación de los valores de las propiedades

  En este proyecto no se ha contemplado el desarrollo de la funcionalidad que permita la modificación de los valores de las propiedades. Sin embargo, esta funcionalidad sería muy útil a la hora de actualizar las propiedades de los elementos que intervienen en un diagrama.

- Modificación de los enlaces, de forma que puedan seguir un camino indicado por el usuario

  La librería Joint JS ha sido utilizada en este proyecto como base para dibujar diagramas. Esta librería dibuja los enlaces entre puertas trazando una línea recta entre ambas, sin tener en cuenta que ya haya dibujado otro elemento en alguna de las coordenadas del trazo.

  Sería interesante mejorar el dibujado de los enlaces para que éstos no se superpusieran a otros elementos, sino que se calculara el camino más corto (por ejemplo, siguiendo una cuadrícula) teniendo en cuenta el espacio del tapiz que está vacío.

7 BIBLIOGRAFÍA

[1] IEEE Power Engineering Society, Substations Committee, Institute of Electrical and Electronics Engineers, American National Standards Institute, y IEEE Standards Board, *IEEE standard definition, specification, and analysis of systems used for supervisory control, data acquisition, and automatic control*. New York, N.Y.: Institute of Electrical and Electronics Engineers, 1994.

[2] «Introducción a Business Process Management (BPM)», 29-abr-2011. [En línea]. Disponible en: http://www.ibm.com/developerworks/ssa/local/websphere/introduccion-bpm/. [Accedido: 30-ago-2015].

[3] «W3C SVG Working Group». [En línea]. Disponible en: http://www.w3.org/Graphics/SVG/. [Accedido: 01-sep-2015].

[4] «Información general sobre ASP.NET MVC». [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/Dd381412(v=VS.108).aspx. [Accedido: 16-ago-2015].

[5] I. Sommerville, V. Campos Olguín, y S. Fuenlabrada Velázquez, *Ingeniería de software*. Madrid: Pearson Educación de México, 2011.

[6] «HTML5». [En línea]. Disponible en: http://www.w3.org/TR/html/. [Accedido: 02-sep-2015].

[7] «Cascading Style Sheets». [En línea]. Disponible en: http://www.w3.org/Style/CSS/. [Accedido: 03-sep-2015].

[8] D. Flanagan, *JavaScript: the definitive guide*, 4th ed. Sebastopol, CA: O'Reilly, 2002.

[9] «Welcome to Ecma International». [En línea]. Disponible en: http://www.ecma-international.org/. [Accedido: 02-sep-2015].

[10] «ECMAScript». [En línea]. Disponible en: http://www.ecmascript.org/. [Accedido: 02-sep-2015].

[11] «JointJS - the HTML 5 JavaScript diagramming library.» [En línea]. Disponible en: http://www.jointjs.com/. [Accedido: 02-sep-2015].

[12] «JointJS - JavaScript diagramming library - Demos». [En línea]. Disponible en: http://resources.jointjs.com/demos/erd. [Accedido: 23-ene-2017].

[13] «Joint JS - API». [En línea]. Disponible en: http://resources.jointjs.com/docs/jointjs/v1.0/joint.html. [Accedido: 23-ene-2017].

[14] «MVC», *The Official Microsoft ASP.NET Site*. [En línea]. Disponible en: http://www.asp.net/mvc. [Accedido: 03-sep-2015].

[15] «Microsoft ASP.NET Site», *The Official Microsoft ASP.NET Site*. [En línea]. Disponible en: https://www.asp.net/. [Accedido: 09-oct-2016].

[16] «Ext.NET - ASP.NET», *Ext.NET*. [En línea]. Disponible en: http://ext.net/. [Accedido: 09-oct-2016].

[17] «SQL Server 2012 | Documentación del producto». [En línea]. Disponible en: https://technet.microsoft.com/es-es/library/hh278298(v=sql.10).aspx. [Accedido: 18-dic-2016].

[18] «SQL Server Management Studio». [En línea]. Disponible en: https://msdn.microsoft.com/en-us/library/ms174173.aspx. [Accedido: 18-dic-2016].

[19] «Microsoft Visual Studio», *Visual Studio*, 13-jun-2016. [En línea]. Disponible en: https://www.visualstudio.com/es/. [Accedido: 09-oct-2016].

[20] «Internet Explorer». [En línea]. Disponible en: https://msdn.microsoft.com/en-us/library/hh772408(v=vs.85).aspx. [Accedido: 01-nov-2016].
[21] «IE - Herramientas de desarrollo F12». [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/gg589500(v=vs.85).aspx. [Accedido: 01-nov-2016].
[22] «Windows y Windows 10 - Microsoft». [En línea]. Disponible en: http://windows.com. [Accedido: 06-sep-2015].
[23] «¿Qué es Linux? | ¡Cámbiate a GNU/Linux!» [En línea]. Disponible en: http://www.getgnulinux.org/es/linux/. [Accedido: 06-sep-2015].
[24] «ISO 25000 STANDARDS». [En línea]. Disponible en: http://iso25000.com/index.php/en/iso-25000-standards#. [Accedido: 06-sep-2015].
[25] «Unified Modeling Language (UML)». [En línea]. Disponible en: http://www.uml.org/. [Accedido: 06-sep-2015].
[26] «Object Management Group». [En línea]. Disponible en: http://www.omg.org/. [Accedido: 06-sep-2015].
[27] «Guía Breve de Servicios Web». [En línea]. Disponible en: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb. [Accedido: 29-ago-2015].