FACULTAD DE CIENCIAS

UNIVERSIDAD DE CANTABRIA Facultad de Ciencias

Proyecto Fin de Carrera

Development of SVG graphics editor for the IDbox platform

Desarrollo de editor de gráficos SVG para la plataforma IDbox

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Jesús Balbontín Fernández
Director: Fernando Vallejo Alonso
Codirector: Miguel Sierra Sánchez

06 - 2013

Índice

Tabla de contenido

Agradecimientos

Quiero expresar con estas líneas mi más sincero agradecimiento a todas las personas que me han apoyado en la realización de este proyecto y a lo largo de mi carrera.

En primer lugar, agradecer a mi familia y a mis amigos más cercanos por aguantarme cuando las cosas no salían tan bien como quería y darme su apoyo incondicional siempre que lo necesitara, en especial a mi pareja que ha estado conmigo hasta las últimas horas antes de terminar esta memoria.

También quiero acordarme de mis amigos de la facultad, sin los que no habría podido superar la carrera, por el apoyo mutuo que nos hemos dado cuando en los momentos bueno y en los malos.

Finalmente agradecer a mi tutor en el proyecto, Fernando Vallejo, por atenderme los últimos días antes de la entrega de la memoria y trasnochar el último día para hacer las últimas revisiones. Además a Miguel Sierra y todo el equipo de CIC por encargarme el proyecto y ayudarme con todo lo que podían en la realización del mismo.

A todos, muchas gracias.

Resumen

CIC Consulting propone el diseño e implementación de un editor de gráficos SVG, específicamente diseñado para acoplarlo con la aplicación denominada IDbox. Dicha aplicación es un sistema de información de planta, que permite a los usuarios obtener datos en tiempo real de señales (como por ejemplo temperatura, presión, etc.), consultar históricos, visualizar graficas de tendencia, entre otras funcionalidades.

La aplicación podrá tanto crear nuevos SVG desde cero, con los controles primitivos de creación de figuras o con la importación de componentes básicos (barras de progreso, gauges, ...), como editar los SVG presentes en la base de datos.

El editor se desarrollará como una aplicación web, de forma que sea accesible desde cualquiera de las opciones móviles implementadas en IDbox.

Palabras clave:

Editor gráfico, SVG, gráficos vectoriales, sistemas de información de planta, IDbox

Abstract

CIC Consulting proposes the design and implementation of a graphics SVG editor, specifically designed for the integration in the application called IDbox. This application is a plant information system that allows users to obtain real-time data signals (such as temperature, pressure, etc.), consulting historical data, seeing trend graphs, among other features.

The application can both create new SVG from scratch, with primitive controls of creating shapes or the importation of basic components (progress bars, gauges, ...) and edit the SVG graphics present in the database.

The editor will be developed as a web application, so that it is accessible from any mobile options implemented in IDBox.

Keywords:

Graphic editor, SVG, vector graphics, plant information system, IDbox

1. Introducción

1.1 Introducción

De una forma u otra, cada vez que se ha realizado el control de un sistema, grande o pequeño, ha sido necesario tener información visual de cómo está funcionando [20]. Así, a medida que los sistemas de control han ido evolucionando y se han hecho cada vez más complejos, ha aumentado también la complejidad de los elementos que proporcionan la información al usuario.

De un simple indicador de aguja, que representa una variable de proceso (por ejemplo, la presión de aire en la instalación neumática), se ha llegado a grandes paneles sinópticos, como el de la Ilustración 1, que muestran el estado de grandes instalaciones (por ejemplo, una central nuclear).

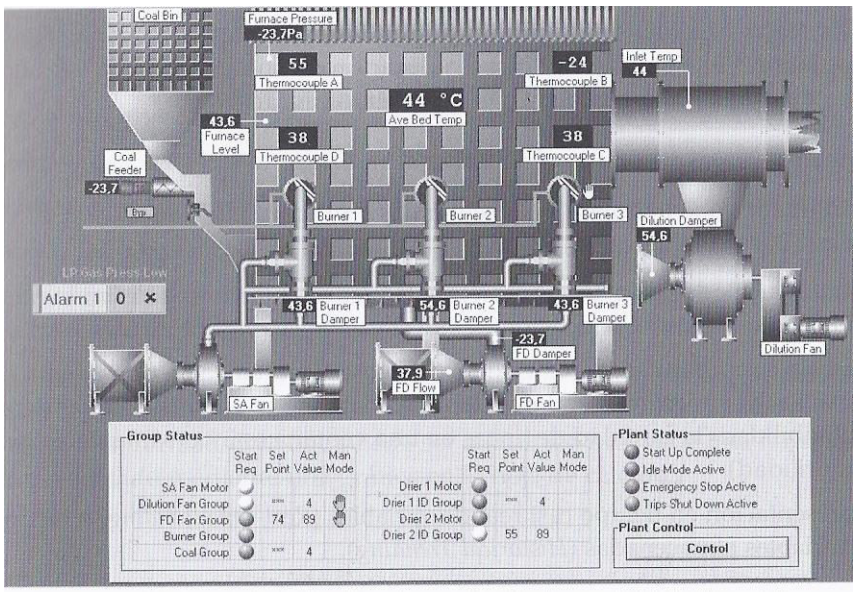

Ilustración 1 - Interfaz SCADA

Si nos ceñimos a la era moderna, las necesidades de ver en la distancia y controlar una máquina aparecen en los primeros cuadros de control, donde multitud de luces indicaban las diferentes situaciones previstas de la máquina. Cualquier situación imprevista, o pasada por alto, podía significar varias horas de trabajo de electricista para llevar la señal olvidada al panel de control y podía ser que no hubiera espacio para colocar el indicador.

La aparición de la informática permitió realizar este tipo de control de manera más sencilla. Ahora ya no sería necesario tener a verdaderos expertos en sistemas de automatización cada vez que hiciera falta cambiar el ajuste de un temporizador en un sistema de control.

Los grandes cuadros de control empezaban a convertirse en monitores que podían mostrar la misma información. Pero cualquier cambio en la presentación era más sencillo de realizar. Bastaban unas modificaciones en el código de la aplicación para en la pantalla apareciera, por ejemplo, una barra de progreso olvidada en el diseño inicial.

La evolución de los sistemas operativos ha incrementado también las posibilidades de estos sistemas, permitiendo las estructuras multipuesto gracias a los sistemas de red informáticos.

Con la irrupción de Internet en el mundo de las comunicaciones industriales ahora es posible conectarse con un sistema de control situado en cualquier lugar del mundo gracias a la tecnología Web-Server: un ordenador dotado de un explorador y la dirección IP del sistema que queremos visualizar serán suficientes.

Damos el nombre de SCADA a cualquier software que permita el acceso a datos remotos de un proceso y permita, utilizando las herramientas de comunicación necesarias en cada caso, el control del mismo.

Los Sistemas de Supervisión de Control y Adquisición de Datos (traducción más o menos aproximada de SCADA, Supervisory Control And Data Acquisition) permiten la gestión y control de cualquier sistema local o remoto gracias a una interfase gráfica que comunica al Usuario con el sistema. En la Ilustración 2 podemos ver el esquema de una arquitectura SCADA.

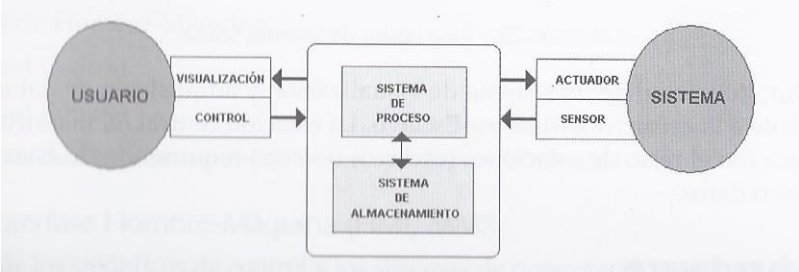

Ilustración 2 - Esquema arquitectura SCADA

En un entorno de este tipo es donde se desarrollará este proyecto, proporcionando una herramienta capaz de producir gráficos que puedan emplearse como sinópticos en una aplicación tipo SCADA.

1.2 Entorno de trabajo

1.2.1 IDbox

Dentro de los sistemas SCADA encontramos IDbox [12]. Aplicación sobre la que debemos dar una visión general antes de empezar a hablar sobre mi proyecto, ya que la aplicación que he desarrollado está ligada a esta plataforma y se pensó como solución a una de las necesidades de la misma.

IDbox es un sistema creado por la empresa CIC-SL que pone a disposición de los usuarios, la información procedente de un conjunto de fuentes de naturaleza heterogénea, de una forma totalmente integrada y homogénea. El sistema es capaz de recoger, integrar, procesar y visualizar grandes volúmenes de información, como por ejemplo señales de una planta industrial, red eléctrica, petroquímica, gas… Tanto centralizados como dispersos geográficamente.

El objetivo final es permitir mediante herramientas analíticas y gráficas, la realización de cálculos y análisis de información (en tiempo real o histórica) y de este modo, favorecer la creación de un escenario de ayuda, tanto en la toma de decisiones, como en la propia operación.

Se ha realizado un esfuerzo importante en la usabilidad y accesibilidad al sistema. De este modo se configuran una serie de clientes que aportan valor en cada situación y en cada momento.

- Cliente web: Cliente de propósito general, accesible mediante web, basado en tecnología AJAX, MVC y SVG. Este cliente permite acceder a IDbox mediante un dispositivo con navegador, típicamente un PC o un Tablet.

- Cliente móvil: Este cliente se ha diseñado para dotar de movilidad a IDbox, permitiendo acceder a la información mediante un dispositivo móvil, tipo Smartphone.

- Cliente Excel: Este cliente integra IDbox en la herramienta Microsoft Excel permitiendo explotar los datos en tiempo real desde el entorno Office.

- Cliente Kinect: El objetivo de este cliente es dotar a IDbox de una interfaz que mediante gestos pueda acceder a la información, posibilitando su uso en entornos críticos donde el riesgo de contaminación es elevado.

En todos los clientes a excepción del Excel se nos proveerá de una serie de sinópticos; gráficos donde se visualizarán de forma representativa diferentes partes del escenario industrial de la empresa, además representarán los datos en tiempo real de forma amigable (mediante textos, barras de progreso y otras animaciones) para dar la sensación de que estamos contemplando el sistema real. Podemos ver uno de los sinópticos de IDbox en la Ilustración 3. Para poder visualizar correctamente estos sinópticos en dispositivos, con diferentes tamaños de pantalla y resolución, éstos están creados en formato SVG.

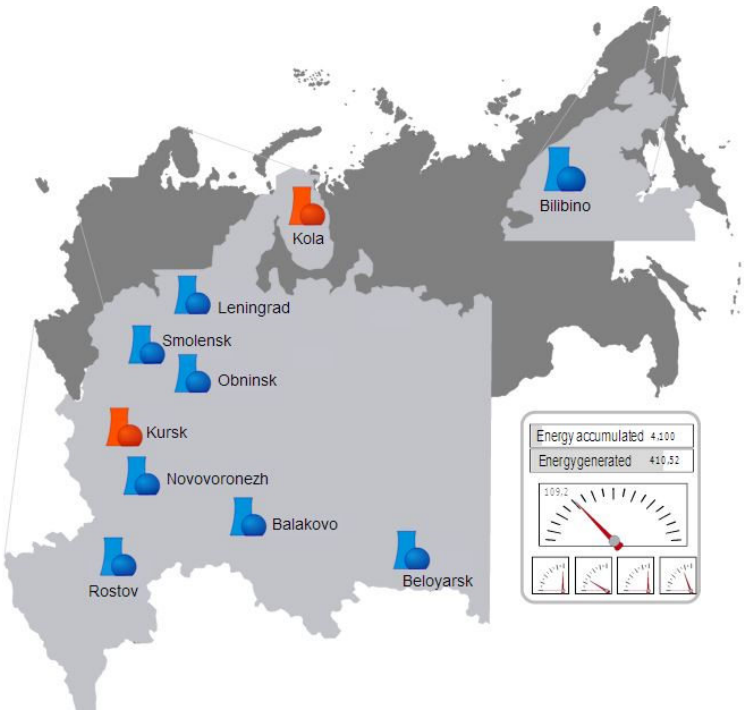

Ilustración 3 - Sinóptico IDbox

1.2.2 Gráficos SVG

SVG [14], que son las siglas de *Scalable Vector Graphics*, es una aplicación de XML que hace posible representar información gráfica de forma portable y compacta. El interés en SVG está creciendo rápidamente, y las herramientas para crear y visualizar archivos SVG ya están disponibles para las grandes empresas. Para comprender las ventajas de los gráficos SVG, comenzaré por dar una descripción de los dos mayores sistemas de gráficos por computador, y describir dónde encaja SVG en el mundo de los gráficos.

- Gráficos rasterizados (Ilustración 4): En los gráficos rasterizados, una imagen es representada como un array rectangular de pixeles. Cada pixel es representado como su valor RGB de color o por un índice en una lista de colores. Estás series de píxeles, también llamadas bitmap, a menudo son almacenados en un formato comprimido. Desde que los dispositivos de visualización modernos son también dispositivos rasterizados, mostrar una imagen requiere un programa de visualización para poco más que descomprimir el bitmap y transferirlo a la pantalla.

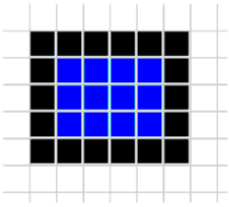

Ilustración 4 - Gráfico rasterizado

- Gráficos vectorizados (Ilustración 5): En un sistema de vectorización gráfica, una imagen es descrita como una serie de formas geométricas. En vez de recibir un conjunto de pixeles final, un programa de visualización vectorial recibe comandos para dibujar formas en coordenadas específicas.

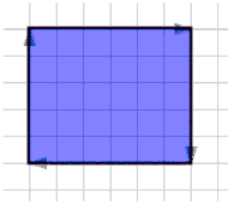

Ilustración 5 - Gráfico vectorizado

Algunas personas describen los gráficos vectoriales como un conjunto de instrucciones para un dibujo, mientras los gráficos bitmap (rasterizados) son puntos de color en lugares específicos. Los gráficos vectoriales entienden lo que son, un cuadrado “sabe” que es un cuadrado y un texto “sabe” que es un texto. Porque ellos son objetos en vez de series de pixeles, los objetos vectoriales puedes cambiar su forma y color, mientras que los gráficos bitmap no. Además, todo texto se puede buscar porque realmente es un texto, no importa lo que parezca o si está rotado o transformado.

A pesar de no ser tan populares como los gráficos rasterizados, los vectoriales tienen una funcionalidad que los hacen indispensables en muchas aplicaciones: pueden ser escalados sin pérdida de calidad de imagen. Podemos ver la diferencia en la Ilustración 6.

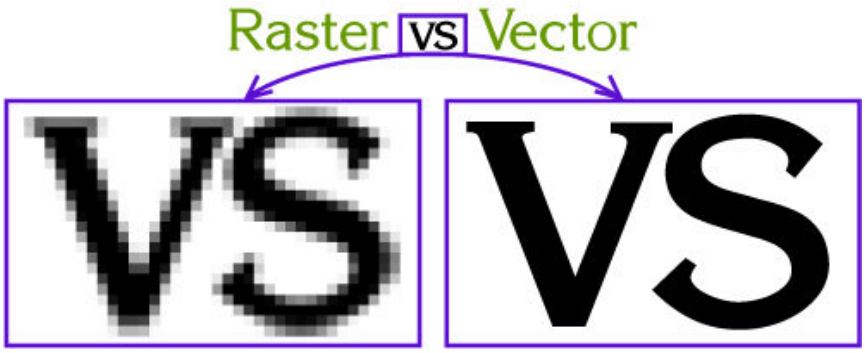

Ilustración 6 – Rasterizado vs Vectorial

Los gráficos vectoriales se usan en:

- Programas de Maquetación Asistida por Computador (CAD), donde la precisión de medida y la habilidad de hacer zoom en un dibujo para ver los detalles es esencial.
- Programas como Adobe Illustrator, los cuales se usan para diseñar gráficos que serán mostrados con visualizadores de alta resolución.
- El lenguaje de descripción de páginas Adobe PostScript, utilizado por muchas impresoras, en el que cada carácter que pintas esta descrito en términos de líneas y curvas.
- El sistema basado en vectores Macromedia Flash para diseñar animaciones, presentaciones y páginas web.
- Cualquier aplicación que necesite redimensionados sin pérdida de calidad de imagen.

Esta última característica es en la que se fijó el equipo de CIC a la hora de elegir el formato en el que se guardarían los sinópticos de IDbox. La redimensión será importante a la hora de hacer zoom en las zonas del sinóptico que requieran atención en determinado momento, y también para el uso de IDbox en el cliente móvil y en el web en el caso de usarlo en una tablet. De los formatos de gráficos vectoriales, el equipo de CIC optó por el formato SVG, por ser actualmente uno de los formatos más ampliamente utilizados.

Debido a la necesidad de crear sinópticos específicos para cada empresa se pensó en la posibilidad de incluir un editor de sinópticos en IDbox, así cada empresa podría crear sus propios escenarios sin necesidad de demasiados conocimientos técnicos.

Ahora nos centraremos un poco más en cómo se construye un gráfico SVG.

Como hemos dicho anteriormente un gráfico SVG es básicamente un documento XML, y al igual que XML está basado en etiquetas. Lo primero que tenemos que hacer para crear un gráfico SVG es escribir su cabecera, que usará la etiqueta <svg>. El elemento raíz <svg> define un ancho (width) y un alto (height) del gráfico final en pixeles. Adicionalmente podemos añadir una etiqueta <title> para ponerle un título al documento, el cual será mostrado como un tooltip si pasamos por encima de gráfico con el ratón.

A partir de aquí lo que debemos hacer para completar el gráfico es ir añadiendo los elementos que le darán forma, a estos elementos se los denomina Formas simples. Los elementos más comunes son los siguientes:

- <line>: El objeto **línea** consta básicamente de dos puntos, **x1 y1** y **x2 y2**, entre los cuales se dibuja un segmento recto.

- <rect>: El objeto **rectángulo** consta básicamente de un punto de origen, **x y**, y las dimensiones del rectángulo, **height y widt**.

- <cricle>: El objeto **círculo** consta básicamente de un punto origen, **cx cy**, y un radio, **r**.

- <ellipse>: El objeto **elipse** consta básicamente de un punto origen, **cx cy**, un radio horizontal, **rx**, y un radio vertical, **ry**.

- <text>: Se puede añadir **texto**, a partir de: un punto base **x** e **y**, tipo de letra **font-family**(serif, garamond, ariel, monospace o courier), tamaño de letras **font-size** y estilo de letra **font-style**(oblique o italic).

- <path>: El objeto **recinto** consta básicamente de puntos que pueden cerrar o no un espacio, estos puntos forman cadenas que se inician con **M**(las mayúsculas tienen el origen en el origen de coordenadas) o con **m**(las minúsculas tienen su origen es el punto anterior).

- <use>: Sirve para hacer duplicados de objetos identificados. Lo explicaré más adelante.

Si queremos tratar un conjunto de objetos como una unidad, SVG nos da la posibilidad de agruparlos, creando un objeto agrupado. Esto se consigue poniendo los susodichos objetos entre las etiquetas de grupo <g> y </g>.

A los objetos se les puede aplicar transformaciones tanto a formas simple como a grupos.

Javascript nos da la posibilidad de acceder a cada uno de estos objetos SVG mediante un modelo de objetos de documento llamado DOM. DOM[19] es esencialmente una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente.

Pensando en la necesidad de crear gráficos SVG que servirían como interfaz de control de procesos industriales CIC me encargó este proyecto: Un sencillo editor de imágenes SVG, diseñado específicamente para la creación de escenarios, con representación de valores y animaciones en tiempo real. Los cuales pueden ser creados y utilizados fácilmente por los empleados.

1.3 Objetivos

Se propone el diseño e implementación de un editor de gráficos SVG, específicamente diseñado para acoplarlo con la plataforma IDbox. El editor se desarrollará como una aplicación web, de forma que sea accesible desde cualquiera de las opciones móviles implementadas en IDbox.

Será capaz de:

- Crear nuevos SVG desde cero, contando con las herramientas de dibujo básicas (trazar líneas, crear formas básicas…), importación de componentes dinámicos (barras de progreso, gauges…) los cuales cambiarán según el valor en tiempo real del proceso al que representan, e importación de formar predefinidas ordenadas por tipo de negocio (nuclear, eólica, petróleo….).
- Editar SVG almacenados en la base de datos del cliente.
- Contará, tanto con un editor gráfico, como con uno de texto. En el editor de texto podremos hacer modificaciones en el código SVG.
- Los objetos prediseñados contarán con atributos especiales dependiendo de qué tipo sean. Los que representen valores en tiempo real contarán con el atributo PID, dependiendo de este atributo los valores se obtendrán de un proceso u otro.
- La aplicación deberá ser soportada por los principales navegadores del mercado.

1.4 Ambiente de desarrollo: Herramientas y métodos

Para su realización se ha usado el entorno de desarrollo Microsoft Visual Studio 2010, ya que es el que la empresa ha usado para los proyectos de desarrollo de IDbox.

Adicionalmente hemos usado Adobe Illustrator e Inkscape para el tratamiento de SVGs cuando teníamos que crear gráficos complejos como las Formas prediseñadas que explicaremos más adelante, pero en contadas ocasiones ya que es un tema del que principalmente se encarga el departamento de diseño.

En cuanto a los lenguajes de programación utilizados, se han utilizado principalmente Javascript y C# bajo el framework .NET.

Para la creación de la interfaz nos hemos ayudado de la librería Javascript Ext.NET, la cual nos facilita la inclusión de botones, paneles, ventanas y demás elementos visiblemente atractivos de forma sencilla.

En cuanto a las pruebas, hemos usado los tres navegadores principales del mercado: Internet Explorer, Google Chrome y Mozilla Firefox; ya que son los tres navegadores a los que IDbox tiene que dar soporte.

Veamos las principales características de las herramientas usadas:

- ***Microsoft Visual Studio [21]***: Es un entorno de desarrollo para Windows, permitiendo desarrollar aplicaciones en varios lenguajes de programación como son Visual C++, Visual C#, Visual J# o Visual Basic .NET.

- ***Illustrator [5]*** e ***Inkscape [6]***: Ambos son editores de gráficos en formato SVG e incluyen funcionalidades como creación de formas básicas, trayectorias, texto, canal alfa, transformaciones, gradientes, edición de nodos, exportación de SVG a PNG, agrupación de elementos, etc. Son los que más funcionalidad aportan y los que son capaces de realizar los gráficos más elaborados, razón por la cual se opta por su uso.

- ***Javascript [19]*** y ***C# [15]***: JavaScript es un lenguaje interpretado, orientado a objetos que se ejecuta normalmente en el lado cliente. Se utiliza principalmente en navegadores web, permitiendo utilizar scripts para alterar el contenido de la página HTML. Mediante la utilización del lenguaje JavaScript el programador puede realizar páginas web en las aparecen animaciones, eventos como pulsar un botón y otras funcionalidades. Al ser un lenguaje del lado del cliente lo hemos usado para las funcionalidades del editor que necesitan interacción con el cliente (que son casi todas), creación de componentes, edición de propiedades…

  C# es un lenguaje de programación cuya sintaxis es muy similar a la de Java. Ambos, C# y Java, son miembros de la familia de lenguajes de programación que derivan de C (C, Objetive C, C++), razón por la cual son similares. Pero la verdad es que muchos de los constructores sintácticos de C# han sido modelados tras varios aspectos de Visual Basic y C++, como la noción de propiedades de clases y parámetros opcionales de VB, o la sobrecarga de operadores de C++. También soporta varias funcionalidades de los lenguajes de programación tradicionales como expresiones lambda y tipos anónimos. Es por esta mezcla de numerosos lenguajes que C# es un producto sintácticamente tan limpio como Java, tan simple como VB, y provee tanta flexibilidad como C++. Se ha usado para funciones que requieren acceso al servidor, como accesos a la base de datos, generación de vistas previas con datos en tiempo real, etc…

- ***DOM (Document Object Model) [19]:*** DOM es una forma de conceptualizar el contenido de un documento. Cuando creamos una página web y la cargamos en un navegador web, el DOM la convierte en algo real. Coge el documento que hemos escrito, usando un código como el de la Ilustración 7, y lo convierte en un objeto.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html;
➥ charset=utf-8" />
    <title>Shopping list</title>
  </head>
  <body>
    <h1>What to buy</h1>
    <p title="a gentle reminder">Don't forget to buy this stuff.</p>
    <ul id="purchases">
      <li>A tin of beans</li>
      <li>Cheese</li>
      <li>Milk</li>
    </ul>
  </body>
</html>
```

Ilustración 7 - Código HTML

Los objetos son contenedores de datos, las variables asociadas a estos objetos las llamamos atributos, mientras que las funciones que pueden ser ejecutadas por el objeto las llamamos métodos. El objeto que maneja el contenido del documento es el *document object.*

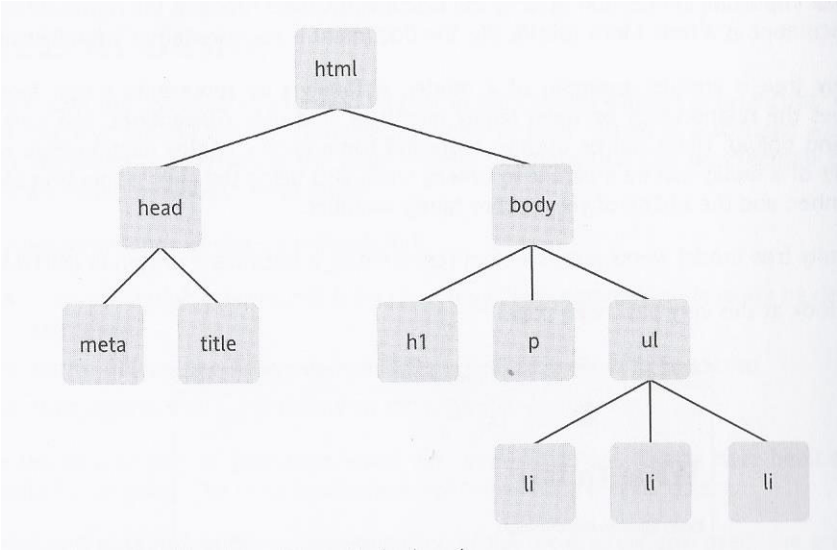

Ilustración 8 - Árbol DOM

Para trabajar con estos objetos también necesitamos un modelo, y es algo que también nos aporta DOM. El modelo de objetos DOM representa el documento como un árbol, o más concretamente como una familia de árboles. Esta familia de árboles describe las relaciones entre los miembros de la familia, y usa convenios, como *parent*, *child* y *sibling*. Podemos usar esto para representar relaciones bastante complejas: un miembro de una familia puede ser padre de otros, mientras a su vez es hijo de otro miembro de la familia, y también hermano de otro. Podemos ver un ejemplo de familia de árboles DOM en la Ilustración 8.

- ***AJAX [18]***: **AJAX**, acrónimo de ***A**synchronous **J**avaScript **A**nd **X**ML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

  Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML. Podemos ver una comunicación típica de Ajax en la Ilustración 9.

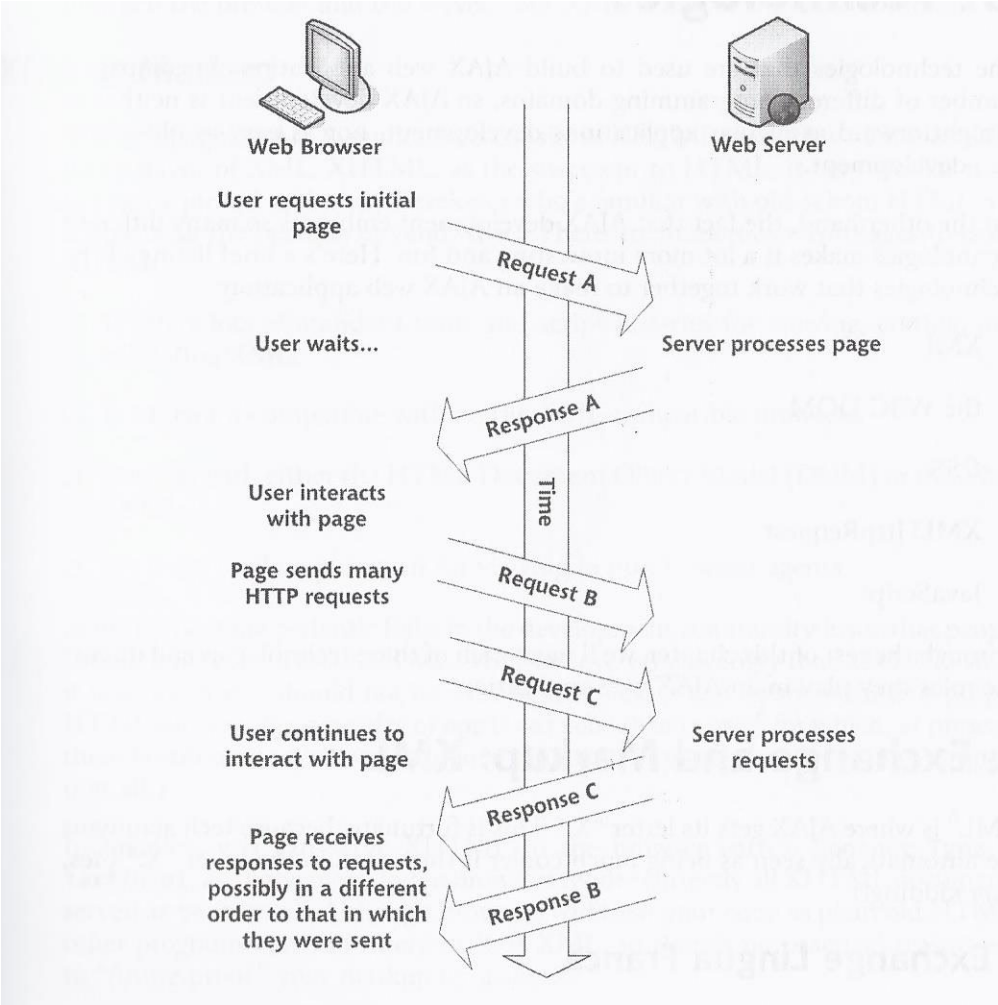

Ilustración 9 - Transacción Ajax

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

- ***.NET [15]***: el framework .NET, que podría considerarse una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de Oracle Corporation y a los diversos framework de desarrollo web basados en PHP. Su propuesta es ofrecer una manera rápida y económica, a la vez que segura y robusta, de desarrollar aplicaciones –o como la misma plataforma las denomina, soluciones– permitiendo una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo. Sus funcionalidades principales son las siguientes:

  - Interoperabilidad con el código existente.
  - Soporte para numerosos lenguajes de programación: C#, Visual Basic .NET, Delphi (Object Pascal), C++, F#, J#, Perl, Pyton, Fortran, Prolog, etc…
  - Un motor común en tiempo de ejecución compartido por todos los lenguajes .NET: Conjunto de tipos bien definidos y entendidos por todos los lenguajes .NET.

- Integración de lenguaje: Soporta herencia a través del lenguaje, manejo de excepciones a través del lenguaje, y depurador de código a través del lenguaje.
- Una exhaustiva librería de clases base: Esta librería ofrece protección frente a la complejidad de las llamadas a las API de bajo nivel y ofrece un modelo de objetos consistente usado por todos los lenguajes .NET.
- Un modelo simplificado de desarrollo: Las librerías .NET no están registradas dentro del registro del sistema. Además .NET permite múltiples versiones de la misma *.dll conviviendo en el sistema.

- ***Ext .NET[16]***: Es un componente avanzado de ASP .NET (WebForms + MVC) integrado en la librería JavaScript Sencha Ext JS. Nos permite construir una rica y moderna aplicación con vanguardistas tecnologías web, inigualable compatibilidad entre navegadores y una avanzada arquitectura MVC. Además nos provee con cientos de demos en su explorador de ejemplos, lo cual es de gran ayuda para desarrolladores que, como yo, no están familiarizados con la herramienta.

1.5 Estado del arte: Editores de gráficos vectoriales (SVG)

La principal razón que motivó a CIC a encargarme este proyecto fue la necesidad de crear un editor de sinópticos completamente web y que estuviera íntegramente conectado con la plataforma IDbox. Vamos a analizar las distintas soluciones que había en ese momento en lo que se refiera a editores SVG.

- **Corel IDRAW Graphics Suite [1]:** Es la principal aplicación de la suite de programas CorelDRAW Graphics, ofrecida por la corporación Corel, y que está diseñada para suplir múltiples necesidades, como el dibujo, la maquetación de páginas para impresión y/o la publicación web, todas incluidas en un mismo programa.
- **Serif DrawPlus [2]**: Es un editor de gráficos vectoriales 2D desarrollado por la compañía Serif. Además de las herramientas tradicionales de dibujo vectorial, DrawPlus ofrece pinceles realistas y naturales que permiten al usuario pintar con acuarelas, aceites y otros medios, conservando la capacidad de edición vectorial.
- **Flatpaint [3]:** Es un diseñador de gráficos online (basado en web) y editor de imágenes gratuito. Flatpaint integra características de publicación de escritorio con herramientas de pintura, dibujo vectorial y productos impresos personalizados en una sola aplicación Flash.
- **Macromedia FreeHand [4]:** El programa fue creado originalmente por la compañía Altsys, y luego licenciado a Aldus. Cuando esta compañía y su cartera de productos fueron adquiridas por Adobe Systems, los nuevos propietarios se vieron obligados a desprenderse de él: FreeHand se situaba en competencia directa con uno de los productos originales más importantes de la empresa (Adobe Illustrator), algo que incluso podía constituir un indicio de prácticas monopolistas. Después de la intervención de la Comisión Federal de Comercio de los EE.UU., el programa volvió a manos de Altsys, que fue comprada posteriormente por Macromedia. Esta firma continuó desarrollando el programa desde la versión 5.5 hasta la MX. Sin embargo, desde 2003 hasta 2006, Macromedia ha mostrado poco interés en el desarrollo del producto. El futuro de FreeHand está abocado a la

desaparición. Hoy en día se continúa vendiendo bajo el mismo nombre de FreeHand MX pero no tiene ninguna nueva función desde su último lanzamiento ni tampoco se exhibe mucho su existencia.

- **Adobe Illustrator [5]:** Es el nombre o marca comercial oficial que recibe uno de los programas más famosos de la casa Adobe, junto con sus hermanos Adobe Photoshop y Adobe Flash, y que se trata esencialmente de una aplicación de creación y manipulación vectorial en forma de taller de arte que trabaja sobre un tablero de dibujo, conocido como "mesa de trabajo" y está destinado a la creación artística de dibujo y pintura para Ilustración (Ilustración como rama del Arte digital aplicado a la Ilustración técnica o el diseño gráfico, entre otros).
- **Inkscape [6]:** Es un editor de gráficos en formato vectoriales SVG, gratuito, libre y multiplataforma. Inkscape tiene como objetivo proporcionar a los usuarios una herramienta libre, de código abierto, de elaboración de gráficos en formato vectorial escalable (SVG) y que cumpla completamente con los estándares XML, SVG y CSS2.
- **OpenOffice Draw [7]:** Es un editor de gráficos vectoriales , y forma parte de la suite ofimática de Apache OpenOffice. Ofrece "conectores" entre otras formas, que están disponibles en una amplia gama de estilos de línea y facilitan la construcción de dibujos como diagramas de flujo . También incluye muchas características que se encuentran en la edición electrónica del software. Los usuarios de Apache OpenOffice también pueden instalar la librería Open Clip Art, que suma una extensa galería de banderas, logotipos, iconos y estandartes para la presentación general y proyectos de dibujo.
- **PhotoLine [8]:** Es un completo editor de gráficos rasterizados para Windows y Mac OS X dirigido al usuario aficionado o profesional avanzado. Sus características incluyen 16 bits de profundidad de color, gestión completa de color, soporte de RGB , soporte de la capa, y manipulador de imágenes no destructivo. Además de eso, también es un editor de gráficos vectoriales y se puede utilizar para la autoedición. Puede importar y exportar todos los formatos comunes de gráficos rasterizados incluyendo formatos RAW de cámaras de imagen, así como archivos PSD, PDF de Photoshop y archivos de animaciones Adobe Flash.
- **sK1 [9]:** Programa de ilustración de código abierto para la plataforma Linux que puede ser utilizado como un sustituto para el software profesional propietario como CorelDRAW o Adobe Illustrator . Las características únicas del proyecto son importadores de formatos CorelDRAW , interfaz con pestañas, motor basado en multi-document Cairo, y gestión de color. El programa fue parado en 2010, y el desarrollo pasó a su sucesor, PrintDesign.
- **Svg-Edit [10]:** Es un editor de gráficos vectoriales on-line y lo que más se acerca a lo que necesitaba la empresa. Sin embargo no cuenta con la posibilidad de guardar los svg creados, solo te genera el código de los mismos.
- **Xara Xtreme [11]:** Es desarrollado por la empresa Xara Group Ltd (Reino Unido). Xara Xtreme comenzó su desarrollo en la década de los 90, la versión original fue creado para el equipo RISC Acorn. El código fuente se encuentra liberado bajo la licencia libre GNU-GPL y los binarios (versión profesional) para windows son de pago. Su interfaz gráfica es muy parecida a CorelDraw, aunque Xara LX presume de mayor rendimiento y estabilidad.

Teniendo en cuenta los programas que había en el mercado en ese momento, una tabla comparativa nos permite saber las características que tiene cada uno y ver si se adecuan con lo

que la empresa estaba buscando. La Tabla 1 nos muestra una comparativa de las características destacables de cada editor, en la Tabla 2 podemos ver los formatos soportados. Se han marcado en verde las características que encajan con los requisitos iniciales del sistema.

Características:

Software	Precio	SO soportados	Tam. página máximo	Zoom máximo	Gestión de color en impresión	Scripting	Es web	Facilidad de uso
CorelIDraw	$ 489	W	45.72 x 45.72 m	45000%	SI	SI	NO	Baja
Serif DrawPlus	$ 99.99	W	¿?	5000%	SI	¿?	NO	Media
Fatpaint	Gratis, $ 29 premium	W/M/U	8000 x 8000 px	Depend on the page size.	NO	NO	SI	Alta
Macromedia FreeHand	$ 399, $ 99 actualización	W/M	5.6388 × 5.6388 m	25600%	SI	SI	NO	Baja
Adobe Illustrator	$ 599, $ 199 actualización	W/M	5.7785 × 5.7785 m	6400%	SI	SI	NO	Muy baja
Inkscape	Gratis (open source)	W/M/U	304.8 × 304.8 km	25600%	NO	SI	NO	Baja
OpenOffice Draw	Gratis (open source)	W/M/U	119 × 119 cm	3000%	NO	SI	NO	Media
PhotoLine	59 €, 29 € actualización	W/M	1,400,000,000 × 1,400,000,000 px	6400%	SI	SI	NO	Media
Sk1	Gratis (open source)	M/U	50 × 50 m	3000%	SI	SI	NO	Media
Svg-edit	Gratis (open source)	W/M/U	¿?	10000%	NO	SI	SI	Alta
Xara Xtreme	Gratis (open source)	U	2.75 × 2.75 m	25601%	SI	NO	NO	Media

Tabla 1- Comparativa editores de gráficos (características)

Formatos soportados para Importar/Exportar

Software	AI	CDR	ODG	PS/EPS	PDF	SVG	SWF	DXF	WMF/EMF

CorelIDraw	SI/SI	SI/SI	NO/NO	SI/SI	SI/SI	Parcial/SI	NO/SI	SI/SI	SI
Serif DrawPlus	SI/NO	NO/NO	NO/NO	SI/NO	SI/SI	SI/SI	NO/SI	SI/SI	SI
Flatpaint	NO/NO	NO/NO	NO/NO	NO/NO	NO/NO	NO/NO	NO/NO	NO/NO	NO/NO
Macromedia FreeHand	SI/Parcial	Parcial/NO	NO/NO	SI/SI	SI/SI	NO/NO	SI/SI	SI/NO	NO/Parcial
Adobe Illustrator	SI/SI	Parcial/NO	NO/NO	SI/SI	SI/SI	SI/Parcial	NO/SI	SI/SI	SI/SI
Inkscape	SI/Parcial	Parcial/Parcial	NO/SI	SI/SI	SI/Parcial	SI/SI	NO/Parcial	SI/SI	SI/SI
OpenOffice Draw	NO/NO	NO/NO	SI/SI	SI/SI	Parcial/SI	Parcial/Parcial	NO/SI	SI/NO	SI/SI
PhotoLine	NO/NO	NO/NO	NO/NO	NO/Parcial	SI/SI	SI/SI	SI/SI	SI/SI	SI/SI
Sk1	SI/SI	SI/NO	NO/NO	SI/SI	NO/SI	SI/SI	NO/NO	SI/NO	SI/SI
Svg-edit	NO/NO	NO/NO	NO/NO	NO/NO	NO/NO	SI/SI	NO/NO	NO/NO	NO/NO
Xara Xtreme	SI/SI	Parcial/NO	NO/NO	SI/SI	SI/SI	SI/SI	NO/SI	NO/NO	SI/SI

Tabla 2 - Comparativa editores gráficos (formatos soportados)

Gracias a este pequeño resumen podemos señalar, si observamos la Tabla 1, que los editores más completos son los de pago, como Illustrator o CorelDraw, sin embargo, además de su precio, tenemos el inconveniente de que no son multiplataforma, la mayoría solo tienen versiones para Mac y Windows. En cuando a los editores “open-source”, el más completo de todos sería Inkscape, pero tiene el inconveniente de que incumple uno de los principales requisitos, no es web. Dicho esto solo nos queda la solución google-code svg-editor, que es el único que cumple con todos los requisitos (el FlatPaint no nos permite exportar ni importar en formato SVG).

Vistas todas las opciones que tenían en el mercado, la empresa optó por encargarme el proyecto. Para hacerlo me basé principalmente en lo que habían hecho los creadores de svg-edit, para posteriormente incluir toda la funcionalidad que necesita IDbox.

2. Análisis de requisitos

2.1 Presentación del problema

Vistos los editores de gráficos disponibles en el mercado, se puede entender la dificultad de los clientes para la creación de sinópticos que representen el estado de los procesos de sus empresas y su posterior importación a IDbox. Es por eso que CIC se vio en la necesidad de crear un editor de sinópticos propio, completamente web y que se pudiera adaptar a las necesidades de negocio de cada cliente.

Dicho esto podemos pasar a enumerar los diferentes requisitos que se me propusieron inicialmente, los que fueron surgiendo a lo largo de las sucesivas fases de desarrollo y la iteración con los clientes se explicarán en el apartado 4 (Implementación).

2.2 Requisitos del sistema

Podemos dividir los requisitos en funcionales y no funcionales.

2.2.1 No Funcionales

- La aplicación será completamente web, sin necesidad de que el cliente instale absolutamente nada en su ordenador.
- La aplicación deberá ser soportada por los tres navegadores principales del mercado: Internet Explorer, Mozilla Firefox y Google Chrome. Los tres en sus respectivas últimas versiones.

2.2.2 Funcionales

- Contará con las siguientes funcionalidades:
  - Herramientas de dibujo básicas:
    - Selección: Permite seleccionar un objeto dentro del lienzo para moverlo o modificarlo. También permite la selección múltiple para agrupar objetos o moverlos juntos.
    - Formas básicas: Basta con pulsar en el lienzo y arrastrar hasta conseguir el tamaño deseado.
      - Rectángulo: Posibilidad de dibujar tanto rectángulos, como cuadrados, como rectángulos a menos alzado
      - Círculo: Posibilidad de dibujar tan elipses, como círculos, como círculos a mano alzada
    - Lápiz: Permite crear líneas a mano alzada.
    - Línea: Permite crear líneas rectas.
    - Path: Permite crear polígonos mediante la creación de series de puntos. La unión de estos puntos formará el polígono.
    - Texto: Permite la introducción de textos.
    - Imagen: Importación de imágenes rasterizadas al lienzo. Funcionalidad inacabada por el momento.
  - Componentes dinámicos: Serán los objetos encargados de realizar las animaciones que muestren el estado de los diferentes puntos de control de la plata industrial a monitorizar. Su representación dependerá de los valores que reciban en tiempo real. Contarán con atributos especiales dependiendo de qué tipo sean. Los que representen

valores en tiempo real contarán con el atributo PID (*Process Identification*), dependiendo de este atributo los valores se obtendrán de un proceso u otro.

- Barra de progreso ( ): Permite mostrar de forma gráfica el estado de desarrollo de una tarea o proceso industrial. Contará con los siguientes atributos:
  - PID: Identificador del proceso al que hace referencia. Podrá ser escogido mediante una ventana que nos mostrará todos los procesos disponibles.
  - Orientación: Podrá ser horizontal o vertical. Crecerá hacia arriba o hacia la derecha dependiendo de la opción que elijamos.
  - Min: Mínimo valor que pueden alcanzar los valores en tiempo real que devuelve el proceso. Si el valor devuelto es igual o menor que este valor, la barra de progreso acabará vacía.
  - Max: Máximo valor que pueden alcanzar los valores en tiempo real que devuelve el proceso. Si el valor devuelto es igual o mayor que este valor, la barra de progreso acabará llena.
  - Dirección: Puede ser inversa o no inversa. Si el valor escogido es inversa, la barra crecerá de derecha a izquierda en el caso de la orientación horizontal y de arriba abajo en el caso de la orientación vertical.

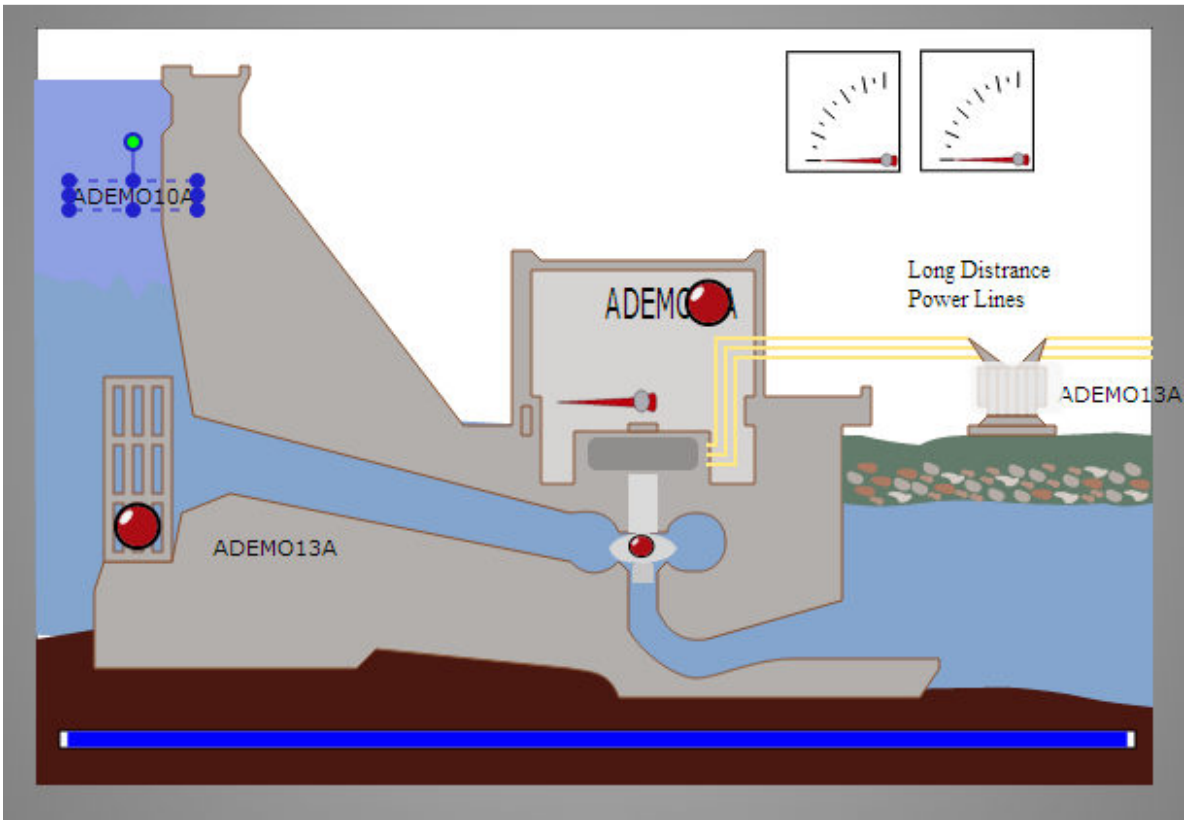

Ilustración 10 - Lienzo

- Gauge (): Aguja que mostrará de forma gráfica, simulando los antiguos indicadores de aguja, el valor de alguna variable del proceso industrial. Podemos ver algunos en la parte superior derecha de la Ilustración 10. Contará con los siguientes atributos:
    - PID: Identificador del proceso al que hace referencia.
    - Min: Lo mismo que con la barra de progreso.
    - Max: Lo mismo que con la barra de progreso.
    - Ang Min: Ángulo que representará el valor mínimo que puede alcanzar el proceso.
    - Ang Max: Ángulo que representará el valor máximo que puede alcanzar el proceso.
- Condicional ( / ): Este componente mostrará un objeto u otro en función del valor recibido y un valor mínimo, que tendrá como atributo:
    - PID: Identificador del proceso al que hace referencia.
    - Valor Mínimo: Si el valor actual del proceso es mayor que éste, se mostrará el segundo objeto, en caso contrario se seguirá mostrando el primero.
- Label (): Mediante los componentes de tipo texto, se mostrarán de forma directa los valores recibidos de algún parámetro de algún proceso industrial. Podemos ver muchos repartidos por el lienzo que representa la Ilustración 10. Contará con los siguientes atributos:
    - PID: Identificador del proceso al que hace referencia.
    - Número decimales: Cantidad de decimales que mostraremos en el texto.
- Link ( ): Rectángulo semi-transparente que contendrá un enlace a otro sinóptico. Atributos:
    - Link: Archivo de la base de datos de sinópticos al que seremos redirigidos en caso de hacer click en el link. Podrá ser escogido a través de una ventana donde se nos mostrarán todos los sinópticos disponibles.

- Formas: Serán gráficos SVG prediseñados y, al igual que los componentes dinámicos, almacenados en una carpeta del servidor. Estarán divididos en paletas de componentes, dependiendo del sector al que pertenezca el cliente: energético, eólico, petrolífero, nuclear…
- Podrá editar sinópticos almacenados en la base de datos del cliente.
- Contará, tanto con un editor gráfico, como con uno de texto.

  En el editor gráfico contaremos con un lienzo donde podremos hacer dibujos e importar distintos componentes prediseñados.

En el editor de texto podremos editar directamente el código XML de los sinópticos, pudiendo resolver las limitaciones que pueda tener el editor gráfico. En la Ilustración 11 vemos el editor de código correspondiente al gráfico de la Ilustración 10.

```
<svg width="575" height="390" xmlns="http://www.w3.org/2000/svg">
 <g>
  <title>Layer 1</title>
  <g id="svg_35">
   <rect class="progress_bar" inverso="0" orientacion="vertical" pid="ADEMO10A" min="0" max="
  </g>
  <g id="layer2">
   <path d="m-1,12611,186.614041575,42.3859610,-132.456151-475,-26.491231-29,-62.052631-12,-3
   <path d="m65,20115,010,5125,010,-515,010,51-5,1010,201115,150125,010,-90110,015,51105,015,
   <path d="m0,31510,751575,010,-40c-11.58514,-1.34055 -23.07758,-2.31033 -35,-5c-10.98212,-2
   <path d="m30,3301135,0110,-101100,10c9.66684,1.9989 20.12244,0.7908 25,151155,0110,-1510,-
   <rect width="35" height="95" x="35" y="180" id="rect5137" stroke-dashoffset="0" stroke-mit
   <rect width="5" height="25" x="40" y="185" id="rect5139" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="50" y="185" id="rect5141" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="60" y="185" id="rect5143" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="40" y="215" id="rect5145" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="50" y="215" id="rect6116" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="60" y="215" id="rect6118" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="40" y="245" id="rect6120" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="50" y="245" id="rect6122" stroke-dashoffset="0" stroke-mite
   <rect width="5" height="25" x="60" y="245" id="rect6124" stroke-dashoffset="0" stroke-mite
   <path d="m255,1251115,010,651-5,010,1515,010,301-23,010,-271-70,010,271-17,010,-451-5,010,
   <text x="448.57144" y="128.57143" id="text6146" xml:space="preserve" font-family="Bitstrea
    <tspan x="448.57144" y="128.57143" id="tspan6148">Long Distrance</tspan>
    <tspan x="448.57144" y="143.57143" id="tspan6150">Power Lines</tspan>
   </text>
   <path d="m250,19515,010,151-5,010,-15z" id="path6168" stroke-dashoffset="0" stroke-miterli
   <path d="m305,23010,30115,010,-301-15,0z" id="path6172" stroke-dashoffset="0" stroke-miter
   <path d="m305,204115,010,41-15,010,-4z" id="path6174" stroke-dashoffset="0" stroke-miterli
   <path d="m309,26010,151-2,210,9111,010,-91-2,-210,-151-7,0z" id="path6176" stroke-dashoffs
   <path d="m312,260c-6,1 -13.93964,1.98236 -17,8c2.99323,9.01431 12,7 18,8c6.07095,-1.18427
   <path d="m415,210110,0c10.38583,2.82417 20.27701,4.16435 30,5c7.51895,-4.1098 16.16373,-4.
   <path d="m480,21010,-5145,010,51-45,0z" id="path6182" stroke-dashoffset="0" stroke-miterli
```

Ilustración 11 - Editor de código

3. Diseño e implementación

3.1 Diseño

3.1.1 Metodología elegida

La metodología elegida ha sido la incremental [17][18]. En muchas situaciones los requisitos iniciales del software están bien definidos en forma razonable, pero el enfoque global del esfuerzo de desarrollo excluye un proceso puramente lineal. Además, quizá haya una necesidad imperiosa de proporcionar de manera rápida un conjunto limitado de funcionalidad al usuario y después refinarla y expandirla en las entregas posteriores del software. En estos casos se elige un modelo de proceso diseñado para producir el software de forma incremental, como es nuestro caso.

Usando esta metodología aplicamos secuencias lineales de forma escalonada conforme avanza el tiempo en el calendario. Cada secuencia se corresponde con el desarrollo de una funcionalidad (herramientas de dibujo, componentes, guardado, etc…). Además usamos esta metodología para conseguir tener pequeñas versiones parcialmente funcionales que podemos ir mostrando a nuestros clientes, para que comprueben que lo que vamos construyendo se corresponden con el que ellos querían en un primer momento, o quieren hacer algún cambio en los requisitos iniciales. Como podemos ver en la Ilustración 12, con cada iteración podemos evaluar el trabajo realizado y desarrollar un plan para el incremento siguiente. El plan afronta la modificación del producto esencial con el fin de satisfacer de mejor manera las necesidades del cliente y la entrega de características y funcionalidades adicionales. Este proceso se repite durante la entrega de cada incremento mientras no se haya elaborado el producto completo.

El proceso incremental, al igual que la construcción de prototipos y otros enfoques evolutivos, es iterativo por naturaleza. Pero a diferencia de la construcción de prototipos, el modelo incremental se enfoca en la entrega de un producto operacional con cada incremento. Los primeros incrementos son funciones incompletas del producto final, pero proporcionan la funcionalidad necesaria que el usuario necesita y una plataforma para evaluarlo.

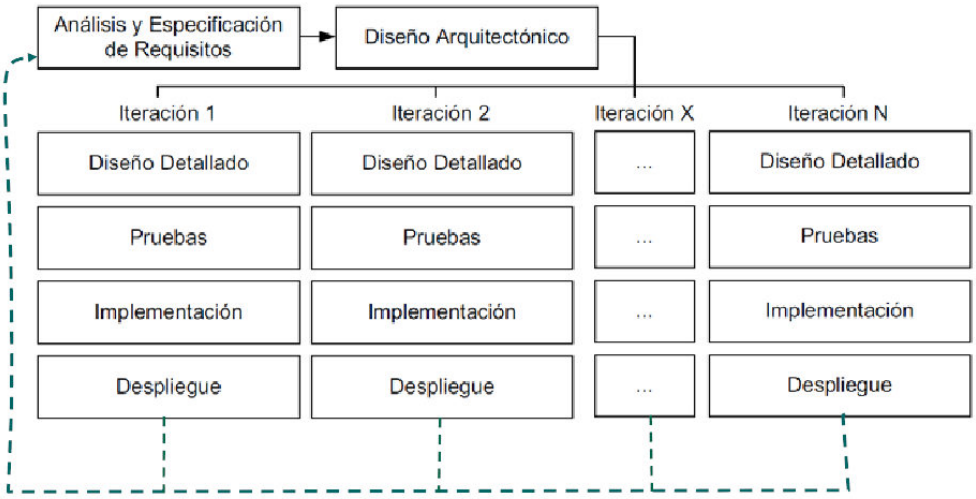

Ilustración 12 - Esquema método iterativo

Para el desarrollo del proyecto se ha elegido este tipo de metodología incremental por la necesidad de los clientes de obtener una funcionalidad básica de edición de sinópticos, para al menos conseguir una vista en tiempo real de los valores de sus procesos.

3.1.2 Patrón de diseño utilizado

Para mi proyecto he seguido el patrón de diseño MVC (Modelo – Vista – Controlador), el cual se ha usado para el desarrollo de IDbox y al que, por consiguiente, he tenido que adecuarme.

El modelo Model-View-Controller (MVC)[13] es un principio de diseño arquitectónico que separa, como vemos en la Ilustración 13, los componentes de una aplicación web. Esta separación ofrece más control sobre las partes individuales de la aplicación, lo cual permite desarrollarlas, modificarlas y probarlas más fácilmente.

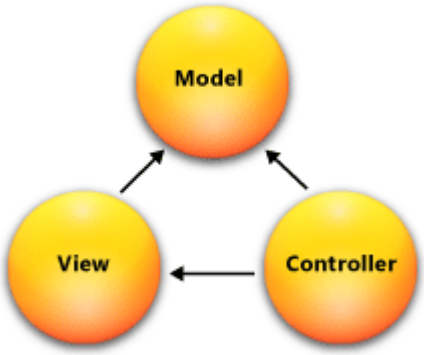

Ilustración 13 - Modelo MVC

El marco de MVC incluye los componentes siguientes:

- Modelos: Los objetos de modelo son las partes de la aplicación que implementan la lógica del dominio de datos de la aplicación. A menudo, los objetos de modelo recuperan y almacenan el estado del modelo en una base de datos. Por ejemplo, un objeto Sinoptics podría recuperar información de una base de datos, trabajar con ella y, a continuación, escribir la información actualizada en una tabla Sinópticos de una base de datos de SQL Server.

  En las aplicaciones pequeñas, el modelo es a menudo una separación conceptual en lugar de física. Por ejemplo, si la aplicación solo lee un conjunto de datos y lo envía a la vista, la aplicación no tiene un nivel de modelo físico y las clases asociadas. En ese caso, el conjunto de datos asume el rol de un objeto de modelo.

- Vistas: Las vistas son los componentes que muestra la interfaz de usuario de la aplicación. Normalmente, esta interfaz de usuario se crea a partir de los datos de modelo. Un ejemplo sería una vista de edición de una tabla Sinópticos que muestra cuadros de texto, listas desplegables y casillas basándose en el estado actual de un objeto Sinoptic.

- Controladores: Los controladores son los componentes que controlan la interacción del usuario, trabajan con el modelo y por último seleccionan una vista para representar la interfaz de usuario. En una aplicación MVC, la vista solo muestra información; el

controlador administra y responde a los datos proporcionados por el usuario y su interacción. Por ejemplo, el controlador administra los valores de la cadena de consulta y pasa estos valores al modelo, que a su vez podría utilizarlos para consultar la base de datos.

El modelo de MVC nos ayuda a crear aplicaciones que separan los aspectos diferentes de la aplicación (lógica de entrada, lógica comercial y lógica de la interfaz de usuario), proporcionando un vago acoplamiento entre estos elementos. El modelo especifica dónde se debería encontrar cada tipo de lógica en la aplicación. La lógica de la interfaz de usuario pertenece a la vista. La lógica de entrada pertenece al controlador. La lógica comercial pertenece al modelo. Esta separación ayuda a administrar la complejidad al compilar una aplicación, ya que nos permite centrarse en cada momento en un único aspecto de la implementación. Por ejemplo, se puede centrar en la vista sin estar condicionado por la lógica comercial.

El acoplamiento entre los tres componentes principales de una aplicación MVC también favorece el desarrollo paralelo. Por ejemplo, un desarrollador de software puede trabajar en la vista, un segundo desarrollador puede ocuparse de la lógica del controlador y un tercero se puede centrar en la lógica comercial del modelo.

3.1.3 Estructura de la aplicación y diagrama de clases

En este apartado voy a explicar un poco cómo está estructurada mi aplicación, para posteriormente pasar a describir cómo ha sido su implementación.

Para empezar, la aplicación deberá contar con un lienzo sobre el cual el usuario pueda realizar los dibujos y modificarlos a su gusto de forma completamente gráfica. Este lienzo estará situado en el panel central de la vista y consistirá en un fondo blanco cuyo tamaño podremos ajustar.

A la izquierda tendremos un panel donde podremos seleccionar las formas básicas que queremos insertar en el lienzo (cuadrados, círculos, componentes dinámicos, etc…), también podremos añadir algunas formas complejas.

Finalmente el panel de la derecha está diseñado para la edición de propiedades de los objetos. Estará oculto al principio y se abrirá cuando seleccionemos un elemento.

Podemos ver el diagrama de clases en la Ilustración 14:

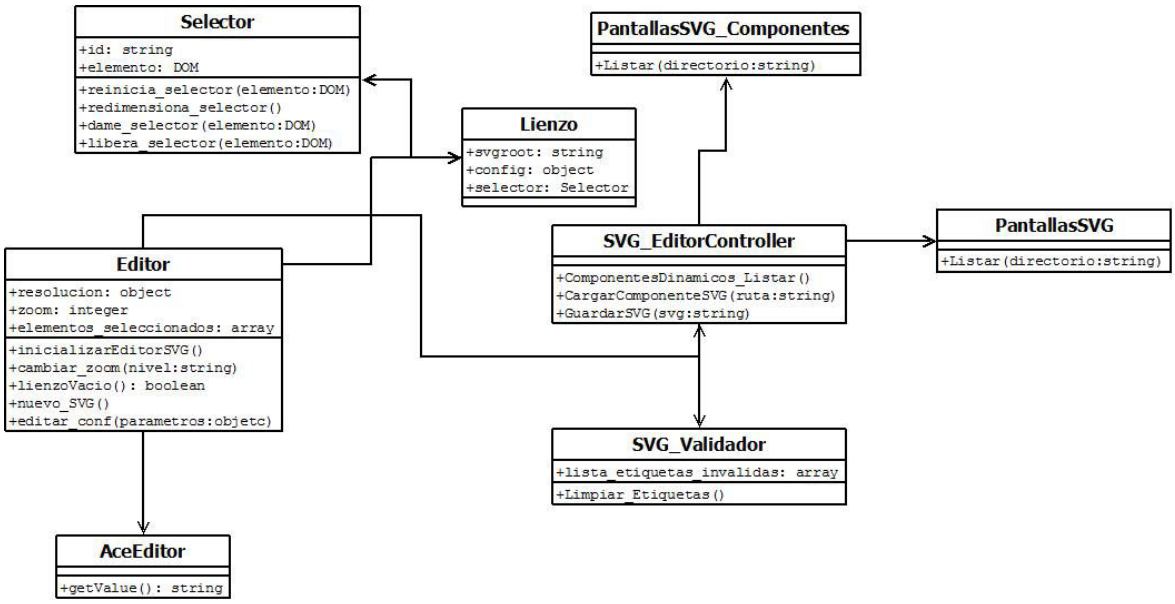

Ilustración 14 - Diagrama de clases

Se han omitido los métodos de la clase Lienzo por ser muy numerosos y emborronarían el diagrama. Se enumerarán en el apartado 4 (Implementación).

Partiendo de esta estructura básica, nos planteamos desarrollar el editor dividiéndolo en las siguientes iteraciones:

- Iteración 1: Se implementarán las funcionalidades de dibujo básicas.
- Iteración 2: Nos centraremos en la creación de componentes dinámicos y formas prediseñadas.
- Iteración 3: Crearemos los cuadros de propiedades y el selector de PIDs.
- Iteración 4: Se implementará la función de guardado de sinóptico.
- Iteración 5: Implementaremos la vista previa.
- Iteración 6: Incluiremos el editor de código SVG.

4. Implementación

En este apartado detallaremos las iteraciones en las que se ha dividido mi proyecto. Como he explicado en el apartado 3.1.1 (Metodología) podemos dividir el trabajo en iteraciones, cada una de ellas consiste en la implementación de una de las funcionalidad descritas en el apartado de requisitos.

4.1 Iteración 1. Desarrollo del lienzo y funcionalidad de las herramientas de dibujo básicas.

En esta primera iteración vamos a crear la funcionalidad básica que podría tener cualquier programa de dibujo: crear líneas y formas básicas, borrar elementos del lienzo, copiar y pegar… Además cuenta con alguna función propia de los gráficos SVG como la posibilidad de agrupar elementos.

Desarrollo:

Se han creado dos fichero de Javascript: lienzo.js y editor.js. Cada uno se corresponde con una clase. La clase Editor será el programa principal y se encargará de interactuar con el usuario realizando todas las funciones del editor llamando a los métodos de las demás clases secundarias.

La clase Lienzo se encarga de implementar las funciones de dibujo que incluyen tanto la creación de formas básicas como la inclusión de textos e imágenes rasterizadas y hasta las funciones de selección, movimiento y transformación de los elementos del lienzo. Tendrá como atributos un elemento HTML llamado *svgroot*, que será nuestro lienzo, y se encargará de contener el gráfico SVG que estamos creando, y un objeto que contendrá los parámetros de configuración del lienzo.

Para la manipulación del elemento raíz (*svgroot*) usaremos el modelo de objetos del documento DOM, el cual hemos comentado anteriormente, ya que nos permite acceder a los elementos de SVG y a sus atributos y manipularlos de manera sencilla. La clase Lienzo, a su vez, utilizará una clase adicional para el manejo del selector de elementos, la clase Selector. Se usará para dibujar un cuadro de selección alrededor del elemento seleccionado. Podemos verlo en la Ilustración 15. Cada objeto selector contará con un identificador de selector y un objeto DOM que corresponderá al elemento que está conteniendo. Contará con las siguientes funciones:

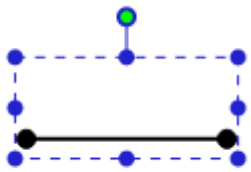

Ilustración 15 - Cuadro de selección

- Reinicia_selector: Recibe como parámetro el objeto DOM asociado al selector. Reiniciará todos los valores del cuadro de selección, es decir, su tamaño y posición.
- Redimensiona_selector: Actualiza las dimensiones del selector para coincidir con las dimensiones del elemento que contiene. Se utilizará cuando cambiemos el tamaño del

elemento seleccionado sin arrastras el cuadro de selección para la redimensión, por ejemplo cuando modifiquemos el tamaño en el panel de propiedades.

- Dame_selector: Recibe como parámetro el objeto DOM correspondiente al elemento cuyo selector queremos obtener.
- Libera_selector: Recibe como parámetro el objeto DOM correspondiente al elemento cuyo selector queremos liberar. Esconde el cuadro de selección.

Las siguientes funciones serán para la creación y manipulación de elementos en el lienzo:

- Copia_elemento: Crea un copia de un elemento dado, actualizando su *id* y el de sus hijos en caso de tenerlos. Recibirá como parámetro el objeto DOM del elemento que queremos copiar. La función consistirá en crear un elemento del mismo tipo, recorrer los atributos del elemento a copiar asignando los mismo valores a la copia y añadir la copia a la lista de hijos del elemento padre de elemento a copiar. Se aplicará la misma función a los hijos del elemento a copiar.
- Dame_elemento: Accede al elemento cuya id nos pasan por parámetro. Devolvemos el elemento usando el método nativo de XML *getElementById*.
- Dame_siguiente_id: Nos devuelve el siguiente id disponible (para que no se repitan entre los elementos). Los identificadores se general a partir del método de Javascript *getTime*, que devuelve la fecha actual en milisegundos.
- Dame_caja_elemento: Devuelve un objeto que tendrá como atributos la posición del elemento pasado por parámetro (*x* e *y*) y su tamaño (*width* y *height*). Recibe como parámetro el objeto DOM del elemento tamaño y posición queremos saber. Utilizamos el método nativo *getBBox* de SVG.

  Las siguientes funciones se encargarán de las transformaciones de los elementos:
- Dame_angulo_rotacion: Devuelve el angulo de rotación del elemento pasado por parámetro. Recibe como parámetros el objeto DOM del elemento cuyo ángulo queremos obtener y un booleano indicando si queremos el ángulo en radianes (será verdadero en ese caso). Para obtener el ángulo buscamos en el atributo *transform* la transformación *rotate*, si es que la tiene, y devolvemos el primer valor, que será el ángulo.
- Pon_angulo_rotacion: Aplica una rotación al elemento pasado por parámetro. Al igual que la función anterior, pasaremos por parámetro el objeto DOM, el nuevo ángulo y un booleano indicando si estamos pasando el ángulo en radianes. Creamos la transformación ("*rotate(angulo, centroX, centroY)*") y se la añadimos al atributo transform del elemento que queremos rotar.
- Dame_lista_transformaciones: Devuelve una lista con todas las transformaciones aplicadas al elemento pasado por parámetro (traslados, rotaciones, escalados y matrices de transformaciones). Pasamos por parámetro el objeto DOM del elemento cuyas transformaciones queremos obtener. Parsearemos el valor del atributo *transform*, guardando cada transformación en un array.

  Las siguientes funciones se encargarán de la selección de elementos. Utilizarán la clase selector.
- Dame_elementos_seleccionados: Devuelve un array con los elementos que están seleccionados en el lienzo. Los elementos seleccionados estarán grabados en una variable global.

- Elimina_seleccion: Elimina la selección actual del lienzo. Obtiene los elementos que están actualmente seleccionados en el lienzo y los deselecciona llamando al método libera selector de la clase selector.
- Anade_seleccion: Añade una lista de elementos a la selección actual. Obtiene la lista de seleccionados actual, añade un nuevo elemento y llama a reinicia selector para que el cambio tenga efecto.
- Quita_elemento_seleccion: Quita el elemento pasado por parámetro de la selección actual. Obtiene la lista de seleccionados actual, elimina el elemento y llama a reinicia selector para que el cambio tenga efecto.
- Borra_elementos_seleccionados: Borra todos los elementos actualmente seleccionados. Recorre la lista de elementos seleccionados y los elimina del lienzo. Usaremos el método nativo de DOM: *elemento.parentNode.removeChild(elemento)*.
- Agrupa_elementos_seleccionados: Mete en el mismo agrupación todos los elementos actualmente seleccionados. Crearemos un nuevo grupo <g> con el método nativo de HTML: *document.createElementNS* (usando *svg* como *namespace*) y añadiremos el contenido del array de elemento seleccionados. Finalmente añadiremos el nuevo grupo al elemento raíz. El resultado será el de la siguiente Ilustración:

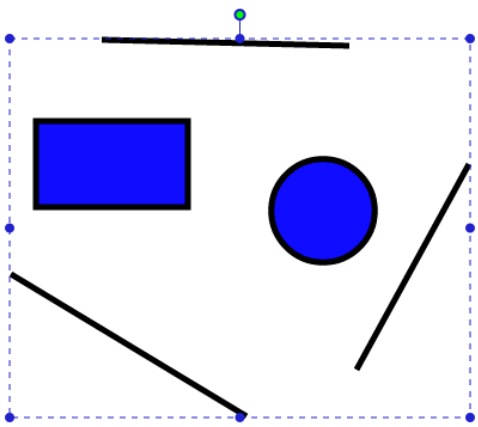

Ilustración 16 - Grupo elementos

- Desagrupa_elementos_seleccionados: Crea una copia de los elementos seleccionados y los coloca como *nextElementSibling* del grupo en el que estaban. Finalmente borra el grupo en el que estaban. El resultado es el de la Ilustración 17.

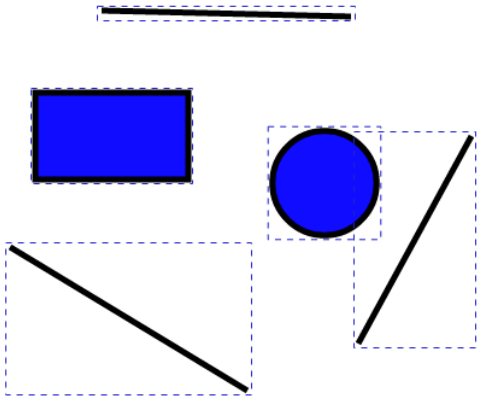

Ilustración 17 - Desagrupación de elementos

- Mueve_elementos_seleccionados: Mueve los elementos seleccionados +/-x posiciones a la derecha y +/-y posiciones arriba. Pasamos como parámetros la distancia recorrida en la dirección x (*dx*) y dirección y (*dy*). Añade una transformación translate: *translate(dx, dy)*.
- Clona_elementos_seleccionados: Copia y recrea todos los elementos actualmente seleccionados. Llama a la función copia_elemento por cada elemento en la lista de elementos seleccionados.

  La dos siguientes funciones sirven para pasar de DOM a string y viceversa.
- svgToString: Devuelve el código XML correspondiente con el SVG que tenemos actualmente en el lienzo. Obtiene el código a partir del atributo innerHTML del objeto DOM del elemento raíz. innerHTML es un atributo nativo de todos los elementos HTML y sus derivados.
- importSvgString: Importa un código XML al dibujo actual.
- Las siguientes funciones sirven para hacer modificaciones en el lienzo:
- Dame_resolucion: Devuelve el tamaño del lienzo. Será el tamaño del gráfico SVG. Se obtiene de los parámetro width y height del objeto DOM del elemento raíz.
- Dame_zoom: Devuelve el estado actual del zoom. El zoom actual estará guardado en una variable global.
- Actualiza_config: Actualiza las opciones de configuración del documento.
- Dame_titulo: Devuelve el título del documento. El título estará guardado en una variable global.
- Cambia_titulo: Cambia el titulo actual del documento por otro pasado como parámetro.
- Cambia_resolucion: Cambia la resolución actual por otra pasada como parámetro. Tiene dos parámetros, x e y. Cambia los atributos width y height del objeto DOM del elemento raíz.
- Cambia_zoom: Cambia el valor del zoom. Añade un viewBox al objeto DOM del elemento raíz. Estará situado en la posición x = 0 e y = 0, y tendrá como tamaño la resolución actual entre el nivel de zoom. En el caso de la Ilustración 18 el zoom está ajustado al tamaño del elemento seleccionado. También llamará al método redimensiona_selector de la clase selector. Finalmente cambiaremos el valor de la variable global que guarda el valor actual del zoom.

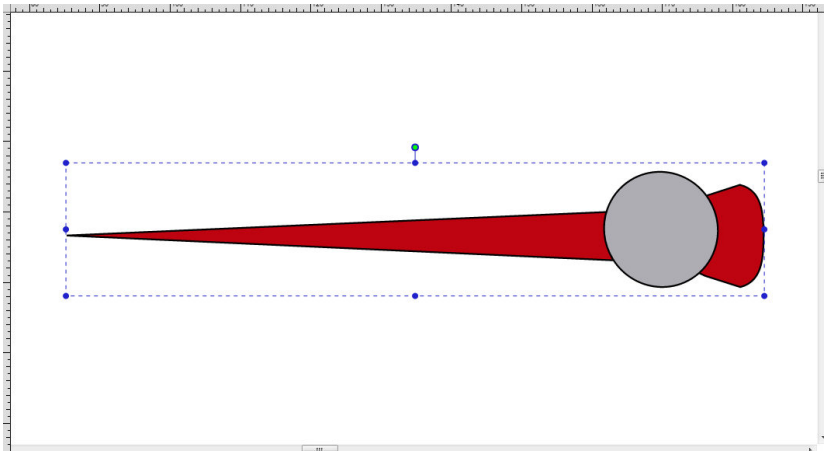

Ilustración 18 - Resultado zoom

- Dame_modo: Devuelve el modo de dibujo actual (rectángulo, círculo, etc…). El modo actual estará guardado en una variable global.

- Actualiza_modo: Cambia el modo de dibujo actual por otro pasado como parámetro.
- Actualiza_lienzo: Actualiza la posición y el tamaño del lienzo después de una operación de zoom.

La clase Editor se encarga de las cuestiones de comunicación entre el usuario y el lienzo. Dota de funcionalidad a los botones de la aplicación llamando tanto a métodos de la clase Lienzo como a funciones del controlador (escritas en C# y usadas desde JavaScript a partir de tecnología Ajax). Además se encarga de almacenar algunas variables globales que representan el estado actual del editor (resolución, nivel de zoom, elementos seleccionados, etc…). En esta iteración solo se implementaron las funciones que llaman a los métodos de la clase Lienzo, ya que para las funciones de dibujo básicas no serán necesarias las llamadas al servidor.
Cuenta con las siguientes funciones:

- InicializarEditorSVG: Creación del objeto lienzo con sus respectivos parámetros de configuración. Además insertará el elemento raíz *svgroot* dentro del contenido del panel central que corresponde con el lienzo. Finalmente se encargará de asignar las siguientes funciones a los eventos de pulsación con el ratón, soltado del ratón y movimiento por el lienzo:
  - mouseDown: Recibe como parámetro un objeto de tipo evento. Tendrá como atributos algunas variables útiles como la posición del ratón. Cuando estamos en modo creación, el elemento será añadido al lienzo con la función *createElementNS* pero su opacidad permanecerá al 10%, como vemos en la Ilustración 19, hasta que soltemos el ratón. Si estamos en modo selección, seleccionamos el elemento y recordamos la posición. En caso de pulsar sobre una de las esquinas del cuadro de selección pasaremos al modo redimensionado.

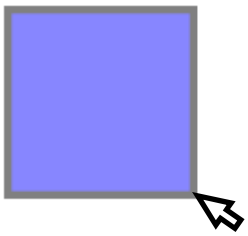

Ilustración 19 - Creación rectángulo

  - mouseUp: Recibe como parámetro un objeto de tipo evento. En el modo creación termina la creación del objeto poniendo la opacidad al 100%.
  - mouseMove: En el modo selección trasladamos temporalmente el elemento aplicando transformaciones de tipo traslado. En el evento mouseUp estas transformaciones serán eliminadas y se reubicará al elemento en su nueva posición. Ya que no todos los elementos tienen los mismos atributos, la forma de crearlos será diferente en algunos caso:
    Para el texto simplemente tendremos en cuenta la posición donde se encontraba el ratón al empezar el movimiento, ya que no cuenta con los atributos de tamaño si no con el tamaño de letra.
    Las líneas tienen cuatro atributos (x1, y1, x2, y2), los dos primeros serán la posición al empezar el movimiento, los otros dos la posición al finalizarlo.

Los rectángulos y los cuadrados se crearán de la misma forma. Los atributos de posición x e y serán la posición del ratón al empezar el movimiento y el width y el height serán la posición final menos la posición inicial en x y la posición final menos la posición inicial en y respectivamente.

Para los círculos tomaremos la posición inicial del ratón como centro y calcularemos el radio restando a la posición final en x la posición inicial en x (nos hubiese dado los mismo hacer en y).

Para las elipses el centro volverá a ser la posición inicial, mientras que también necesitaremos dos radios, el radio en y lo sacaremos de restar la posición final en y menos la inicial en y, el radio en x lo sacaremos restando la posición final en x menos la posición inicial en x.

- addSVG: Importa el contenido de un SVG pasado como parámetro, definiendo el código en un <def> y creando un nuevo <use> que le haga referencia. Se creará un elemento de tipo <use> al que asignaremos un nuevo identificador, en caso de no existir se creará un objeto <symbol> (que es el que contiene todos los elementos <def>) y un <def> que contendrá el código SVG del nuevo SVG. Se utilizará para importar formas y editar los sinópticos guardados en la base de datos.
- loadFromString: Cambia el contenido del elemento raíz por el código SVG pasado por parámetro.cambiarModo: Pasamos de un modo a otro en función de una variable pasada por parámetro.
    - Selector: En este modo podremos seleccionar objetos del lienzo, moverlos, transformarlos y agruparlos.
    - Rectángulo, círculo, texto, path, línea, lápiz, imagen: Desde estos modos podremos crear las formas de dibujo básicas.
    - Zoom: Podremos ajustar la escala de la vista del lienzo.
- cambiarZoom: Ajustamos el zoom de forma automática eligiendo entre tres opciones: ajustar al lienzo, ajustar a la selección y ajustar a todo.
- lienzoVacio: Comprueba los elementos que tenemos en el lienzo y devuelve true en caso de no encontrar ninguno, false en caso de encontrar uno o más. Mira si el elemento raíz *svgroot* tiene algún hijo, en caso de no tener significará que el lienzo está vacío.
- nuevo_SVG: Borra el contenido del lienzo. Tras la implementación de la funcionalidad de guardar se pide una confirmación en caso de haber cambios sin guardar. Se mostrará una ventana tal y como vemos en la Ilustración 20. Elimina todos los hijos del elemento raíz.

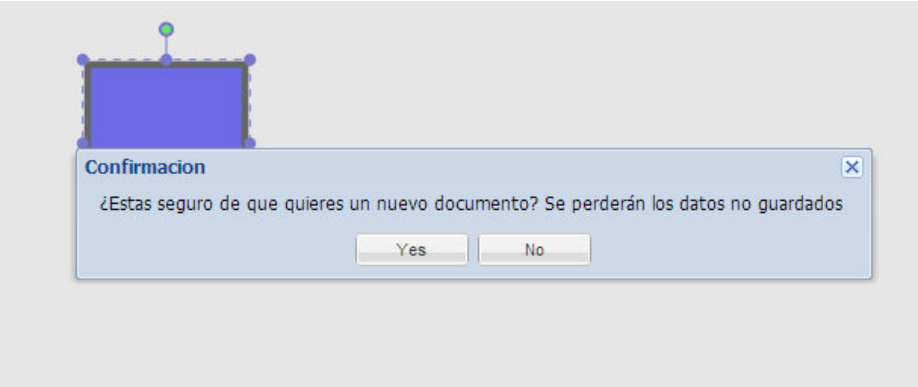

Ilustración 20 - Ventana confirmación

- editar_conf: Muestra una ventana, como la que vemos en la Ilustración 21, para editar las propiedades del documento, título y tamaño del lienzo.

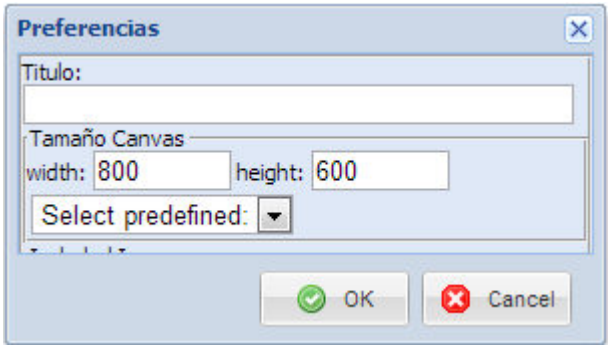

Ilustración 21 - Ventana configuración

- eliminar_seleccion: Elimina el elemento seleccionado en el lienzo, o el grupo de elementos en caso de haber una agrupación. Usará el método borrar_elementos_seleccionados de la clase Lienzo.
- clonar: Copia el elemento seleccionado y crea uno idéntico asignándolo un nuevo identificador. Hace uso de la función clona_elementos_seleccionados de la clase Lienzo.
- mandar_al_frente y mandar_al_fondo: Mueve el elemento a la cima o a la base de la pila de elementos dependiendo del botón pulsado. Elimina el elemento seleccionado y lo pone como nextElementSibling (próximo hermano) del elemento más profundo en caso de mandar al fondo o como previousElementSibling (anterior hermano) en caso de mandar al frente.
- agrupar: Mete los elementos seleccionados en el mismo grupo. Usa el método agrupa_elementos_seleccionados de la clase Lienzo.
- desagrupar: Elimina el grupo y coloca los elementos que contenía por separado en el lienzo.

Además de los dos ficheros Javascript que implementan la funcionalidad básica del editor se crea en esta iteración la vista principal de la aplicación Editor.aspx. Ésta incluye tres paneles:

- Panel Central: Incluye el lienzo y un ToolBar en la parte superior donde se encuentran los botones de nuevo documento, configuración, eliminar, clonar, mover (frente o fondo) y agrupar/desagrupar. Podemos verlo en la Ilustración 22.

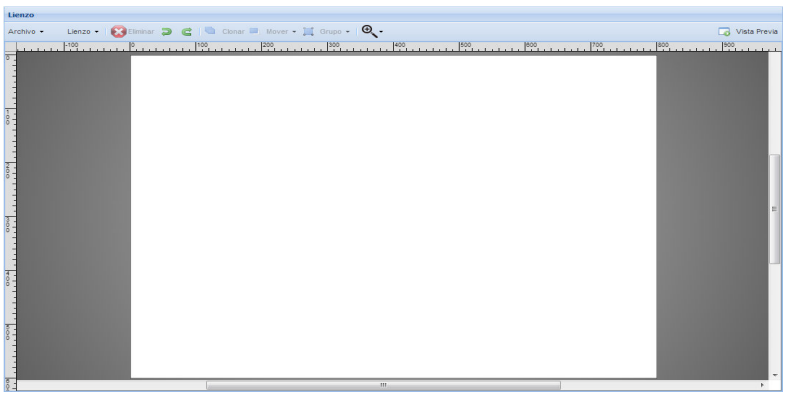

Ilustración 22 - Panel Central

- Panel Oeste: Como vemos en la Ilustración 23, incluye los botones encargados de los cambios de modo y el zoom.

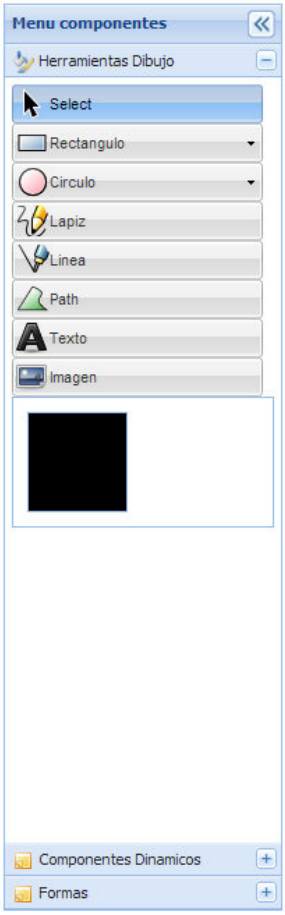

Ilustración 23 - Menú herramientas

- Panel Este: Reservado para la edición de propiedades.

Completada la iteración se pasó a su respectiva fase de pruebas. Consistieron en lo siguiente:

- Prueba 1: Abrir editor. Se comprobó con el depurador que todos los Javascript se habían cargado correctamente y que la clase principal Editor creado correctamente el objeto de tipo Lienzo con los atributos correctos.
- Prueba 2: Una vez creado el lienzo se pasó a probar la creación de elementos. Se crearon un rectángulo, un círculo, una elipse, algunas líneas y un texto.
- Prueba 3: Probamos los métodos de manipulación de elementos realizando copiados y pegados, agrupación y alguna transformación con varios elementos de los que estaban en el lienzo.
- Prueba 4: Hicimos también algunas pruebas con el zoom y la ventana de configuración. Probamos con cinco tamaños de lienzo diferentes: 640x480, 800x600, 1024x768, 1280x960 y 1600x1200.
- Prueba 5: Cerramos el editor. Comprobamos que las demás funciones de IDbox seguían operativas.

4.2 Iteración 2. Inclusión de componentes dinámicos y formas.

Para esta iteración hemos creado dos nuevos sub-paneles en el Panel Oeste, el que vemos en la parte izquierda de la Ilustración 24 mostrará una lista de los componentes dinámicos que podemos incluir en el lienzo, el de la parte derecha de la Ilustración 24 tendrá una lista algo más extensa, y nos permitirá seleccionar gráficos prediseñados, los cuales podrán ser seleccionados por las empresas para que se adecuen a sus procesos de negocio.

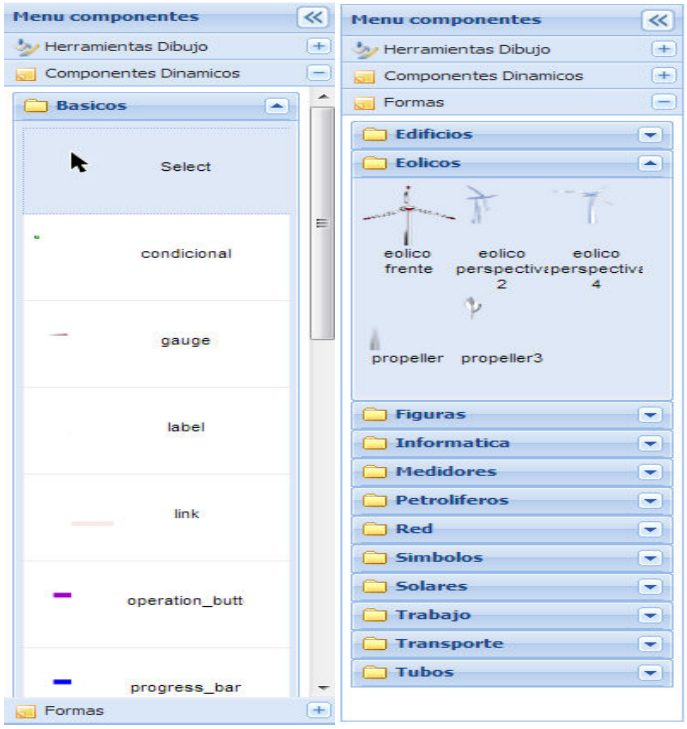

Ilustración 24 - Menú componentes dinámicos (izquierda) formas (derecha)

Lo primero que hacemos para incluir un nuevo componente en la aplicación es crear su código SVG. Este código se colocará en una capeta común para todos los componentes, la cual será cargada de forma dinámica mediante JavaScript y mostrada en un panel desde el cual podremos cargar el componente deseado.

Desarrollo: Para el desarrollo de esta iteración se han creado dos nuevos archivos Javascript, menuComponentesDinamicos.js y menuComponentes.js.

menuComponentesDinamicos.js se encarga de la generación dinámica del contenido del panel de componentes dinámicos, además de implementar su funcionalidad. Cuenta con las siguientes funciones:

- ComponentesDinamicos_Cargar: Accede a la lista de componentes dinámicos ubicada en una carpeta del servidor, la cual será pasada como parámetro, y llama a la función *cargarComponentesDinamicos* pasando como parámetro la lista de componentes. Para acceder a la carpeta debemos acceder al servidor, por lo que deberemos hacer una llamada al controlador mediante una petición Ajax a la función C# *ComponentesDinamicos_Listar*.

- ComponentesDinamicos_Listar: Hace una llamada al Modelo, en concreto a la clase PantallasSVG_Componentes y hace uso de su función Listar a la que pasamos el directorio que queremos listar y un booleano indicando si es un componentes dinámico. La clase PantallasSVG_Componentes no la he implementado yo, ya que no se me ha permitido hacer nada relacionado, por lo cual no comentaré su funcionamiento.

- cargarComponentesDinamicos: Crea tantos directorios como sub-carpetas tenga la carpeta de componentes dinámicos. Dentro de estos directorios creamos tantos archivos de componentes dinámicos como archivos tenga la sub-carpeta.
- crearCarpetaDinamico: Crea un nuevo panel en el sub-panel de componentes dinámicos por cada carpeta en el directorio de componentes dinámicos. Podemos ver los dos sub-paneles que se crean en el menú de componentes dinámicos en la Ilustración 25.

Ilustración 25 - Carpetas dinámicas

- creaArchivoDinamico: Se crea un panel de tipo cuadrícula con una fila por cada archivo que contenga la carpeta. Podemos verlo en la siguiente Ilustración:

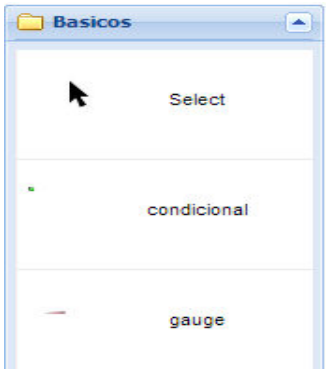

Ilustración 26 - Archivos dinámicos

- ComponenteDinamicoSleccionar_Click: Llama al controlador para acceder al contenido del archivo seleccionado (que contendrá el código SVG del componente) mediante la función C# CargarComponenteSVG. Esta función devolverá el código SVG del componente, el cual insertaremos en el lienzo mediante la función de la clase Lienzo importSvgString. Finalmente llamaremos a la funciónactualiza_lienzo, también de la clase Lienzo.
  - CargarComponenteSVG: Primero comprueba, por seguridad, que la ruta que pasamos por parámetro se corresponde con el directorio de componentes dinámicos ("/Content/DOCUMENTOS/SVG_Componentes/"). Finalmente obtiene el contenido del fichero en formato texto mediante la función *System.IO.File.ReadAllText(rutaFisica)*. Devuelve el texto en un string en caso de tener éxito, “NP (No tiene permisos)” en caso contrario.

menuComponentes.js tiene una funcionalidad muy parecida al anterior script, sin embargo lo que hace es importar SVGs enteros que añaden mucho código al dibujo existente. Por eso

decidimos hacer uso de las etiquetas SVG <use> y <def>, que sirven para replicar el mismo dibujo sin necesidad de replicar todo el código. La etiqueta <use> será la que se replique, y hará referencia a la etiqueta <def>, donde estará el código del dibujo. Cuenta con las siguientes funciones:

- Componentes_Cargar: Accede al listado de componentes a través de una llamada al controlador, al igual que con los componentes dinámicos. Pasamos la lista a la función cargarComponentes.
- cargarComponentes: Crea tantos directorios como sub-carpetas tenga la carpeta de componentes. Dentro de estos directorios creamos tantos archivos de componentes como archivos tenga la sub-carpeta.
- crearCarpetaDinamico: Crea un nuevo panel en el sub-panel de componentes.
- creaArchivoDinamico: Se crea un panel de tipo cuadrícula con una fila por cada archivo que contenga la carpeta.
- ComponenteSleccionar_Click: Llama al controlador para acceder al contenido del archivo seleccionado (que contendrá el código SVG del componente) mediante la función C# CargarComponenteSVG (la misma que usamos con los componentes dinámicos). Esta función devolverá el código SVG del componente, el cual insertaremos en el lienzo mediante la función de la clase Editor addSVG.

Al ser cargados dinámicamente la mayoría de los paneles necesarios para esta funcionalidad, no se han hecho demasiadas modificaciones en la vista principal. Únicamente se han añadido dos sub-paneles nuevos en el Panel Oeste, que acompañan al de herramientas de dibujo básicas, uno para los componentes dinámicos y otro para las formas.

Las pruebas de esta iteración son bastante sencillas, basta con añadir todos los componentes dinámicos y formas que tenemos en el directorio de componentes.

- Prueba 1 (Componentes dinámicos): No tuvimos demasiados problemas ya que en ese momento solo contábamos con cinco componentes, los cuales no eran gráficos complejos, la mayoría eran formas básicas con algunos atributos especiales.
- Prueba 2 (Formas): Aquí nos encontramos con algunos problemas a la hora de importar dibujos complejos. La mayoría de estos gráficos prediseñados eran creados por el equipo de diseño mediantes programas de diseño como Adobe Illustrator, los cuales algunas veces embebían etiquetas especiales de su programa que no eran reconocidas por el modelo de objetos DOM, y por lo tanto hacían fallar a nuestro editor. El único remedio que se nos ocurrió, a parte de “limpiar” los SVG nosotros mismos, fue crear una función de C# que examinara el contenido de los gráficos antes de importarlo y eliminara las etiquetas inválidas.

Esta función de validación la crearemos dentro de una nueva clase llamada SVG_Validador, que tendrá como único atributo una lista de etiquetas a retirar. La función de validación buscará cada una de estas etiquetas a retirar con el método *indexOf* de la clase String y las eliminará con el método *Remove* también de la clase String.

4.3 Iteración 3. Cuadros de propiedades y selector de PIDs

Tras conseguir que nuestro editor pudiera dibujar formas básicas e importar gráficos prediseñados, llegó la hora de poder editar sus propiedades gráficamente. Ya que tenemos mucho código en la vista principal decidimos usar un control de usuario a parte, para tenerlo todo mejor ordenado y accesible. Podemos ver el panel de propiedades en la Ilustración 27.

Un control de usuario o *user control (uc)* es una herramienta que nos ofrece Visual Studio. Es un fragmento de una vista, podríamos decir que es una vista en miniatura que podemos usar en varias partes de la vista principal, para no repetir código y tenerlo todo más ordenado.

Tras su creación con sus respectivos paneles y formularios pasamos a programar su funcionalidad, para ello se crea una función en JavaScript dentro de la clase Editor que se active en caso de que uno de los registro pierda el foco o se pulse la tecla *enter*. Esta función accederá a las propiedades del elemento que esté seleccionado en ese momento en el lienzo y dependiendo del registro que ha provocado el evento se actualiza una propiedad u otra.

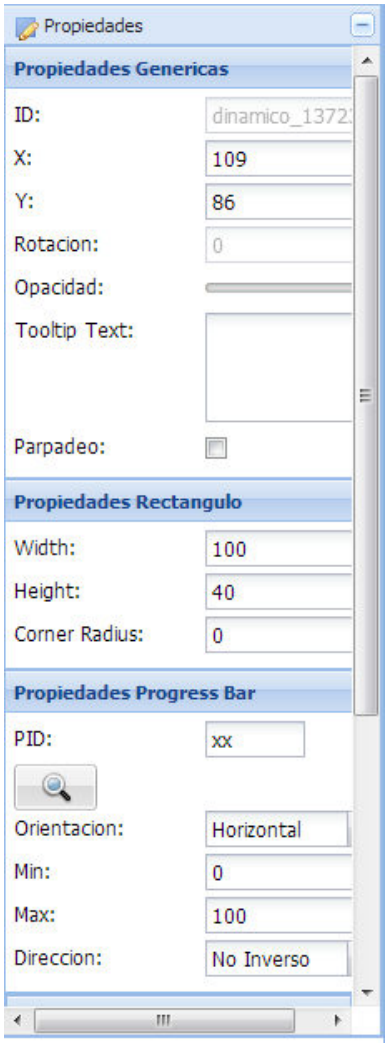

Ilustración 27 - Panel propiedades

Desarrollo: Además de la función antes mencionada de actualización de propiedades, también debemos crear otra función para cargar las propiedades del elemento seleccionado en el Panel Este (el cual habíamos reservado previamente para las propiedades). Esta función mostrará

unos registros u otros del nuevo uc (que estará incrustado dentro del Panel Este) dependiendo del elemento seleccionado. También la añadiremos a la clase Editor.

- Cargar_propiedades: Lo primero que hace esta función es cargar las propiedades genéricas, que serán las que tengan todos los elementos que podemos dibujar (x, y, rotación, opacidad, etc…). Después pasamos a las propiedades específicas de cada elemento, para mostrar unos registros u otros hacemos una comprobación del atributo *tagName* del elemento. Puede ser *rect*, *line*, *text*, *g*, etc… Finalmente comprobaremos si se trata de un componente dinámico haciendo una comprobación del atributo *className*, que puede ser: *progress_bar*, *conditional*, *label*, etc… De esta forma podremos cargar también sus atributos especiales. Algo que no se ha dicho antes pero debemos tener en cuenta es que distinguimos a los elemento que tienen el distintivo de ser dinámicos por el atributo *class*, en este atributo pondremos el tipo de componente dinámico que será dicho elemento.
- Actualizar_propiedades: Como he dicho anteriormente esta función se activa con la pérdida del foco de algunos de los registros del *uc* o la pulsación de la tecla *enter* y tiene como parámetro la propiedad a actualizar. Está implementado mediante un *switch* que actualiza determinada propiedad del elemento seleccionado en función del parámetro recibido. Accedemos al elemento seleccionado que estará guardado en una de las variables globales de la clase Editor. Modificaremos el atributo en cuestión mediante el método *setAttribute* nativo de DOM.

La fase de pruebas en este caso fue bastante tediosa. Debíamos comprobar que los cuadros de propiedades se mostraban siempre correctamente y que al modificarlos, los cambios quedaban correctamente plasmados en el lienzo. Además de esto debíamos hacerlo para cada uno de los elementos que la herramienta nos permitía crear, tanto formas básicas como componentes dinámicos, ya que cada uno cuenta con diferentes atributos. Por fortuna, no fue necesario probarlo con cada una de las formas prediseñadas ya que, al crearse mediante elementos <use>, solo cuentan con los atributos *x*, *y*, *width* y *height*.

Hay uno de los atributos que se maneja de forma especial, es el PID. Para él hemos creado una ventana de selección, la cual nos muestra un listado de todos los procesos industriales que tenemos almacenados en la base de datos y nos permite seleccionar uno de ellos.

4.4 Iteración 4. Funcionalidad de guardado.

Completada la funcionalidad básica del editor de gráficos, tenemos que empezar el acoplmiento con idBox. Para eso necesitamos hacer llamadas a los controladores, a las demás vistas del proyecto e incluso conexiones con la base de datos. Así que a partir de aquí algunas de las funciones han sido desarrolladas en C# en lugar de Javascript.

Llegados a este punto del proceso de trabajo está claro que nos falta algo fundamental, el guardado de los sinópticos creados, sin el cual todo el trabajo realizado no serviría de nada. Ésta es una de las funcionalidades que necesita acceso a la base de datos por lo cual se ha usado, además de JavaScript, C# para su implementación.

Desarrollo: Llamaremos a la función de guardar a través de un botón colocado en el ToolBar superior, encima del lienzo. La funcionalidad de este botón se implementa mediante dos funciones, una Javascript y otra C#.

- **GuardarArchivoSVG**: Se encarga de llamar al controlador pasando como parámetros el nombre del archivo y el string con el código SVG. El controlador hará uso de la función *GuardarSVG*. Si tiene éxito desactivaremos el botón de guardar y mostraremos una notificación de éxito en la operación. En caso contrario mostraremos una notificación de fallo en la operación.
- **GuardarSVG**: Esta función simplemente pasa el código svg en formato string a array de bytes. Tras esto haremos una llamada a una función del modelo pasándole este array. Esta función ya había sido anteriormente implementada para idBox, por lo que solo tuve que hacer la llamada a la susodicha función (*Models.PantallaSVG.GuardarSVG*).

Además del botón de guardado hemos tenido que implementar una ventana de tipo Ext .NET para introducir el nombre del archivo y el título del sinóptico.

Adicionalmente se pensó en la posibilidad de guardar los SVG en formato comprimido SVGZ. Sin embargo esto daba problemas a la hora de visualizarlos en el visor, por lo que se dejó de lado por el momento.

La fase de pruebas consistió en el guardado de varios sinópticos y la comprobación de que las funciones y las llamadas al modelo funcionaban perfectamente, sin ningún problema de permiso ni de otro tipo.

Llegados a este punto pudimos pasar a una fase beta de despliegue. Con las funcionalidades existentes hasta el momento nuestros clientes deberían ser capaces de crear sus propios sinópticos y guardarlos en sus equipos. A partir de este momento empezamos a recibir sugerencias y valoraciones sobre nuestro editor por parte de algunos de nuestros clientes. Esta fase beta del diseñador se añadió a la versión de IDbox instalada en el servidor de desarrollo de CIC.

4.5 Iteración 5. Vista previa.

Una vez completada toda esta fase previa, a uno de nuestros clientes se le ocurrió la posibilidad de acceder a una vista previa durante la edición de los sinópticos, para evitar la tediosa tarea de realizar un guardado y acceder a la lista de sinópticos cada vez que queremos comprobar si nuestro trabajo está quedando como nosotros queremos.

Desarrollo: Para realizar la vista previa tenemos que hacer uso de una vista ajena al editor, esta vez la del Visor de sinópticos. Para llamar al visor necesitaremos, primero que el sinóptico esté almacenado en la base de datos (temporalmente al menos), y segundo su identificador para que el visor sepa qué mostrar. El guardado del sinóptico lo haremos como con el guardado normal, mediante una llamada asíncrona Ajax al controlador a través de una función Javascript a la clase Editor.

- **Vista_previa**: Lo primero que hace es llamar al controlador para guardar un archivo temporal, ya que el controlador del visor solo muestra sinópticos almacenados en base de

datos. El controlador devolverá el identificador del archivo temporal, que es el que tenemos que pasar a la vista del visor para que funcione. Finalmente se creará una ventana emergente, como la que vemos en la Ilustración 28, a la que como contenido le pasaremos la dirección del visor de sinópticos.

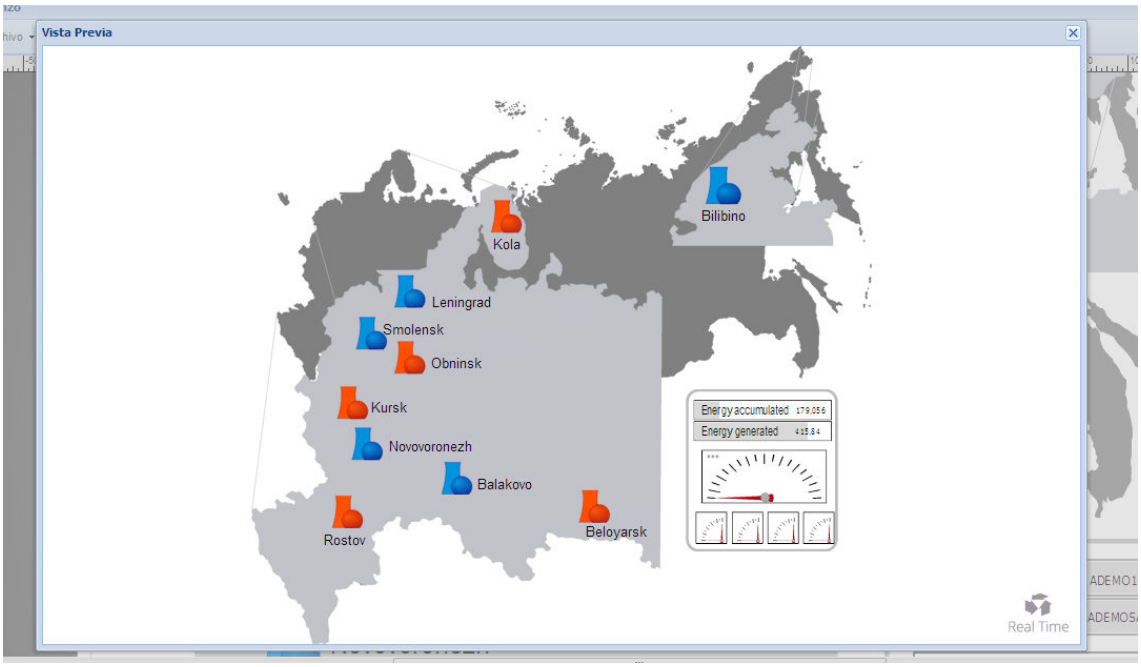

Ilustración 28 - Vista previa

- **GuardarSVGTemporal**: Generará la ruta de guardado temporal y el nombre del archivo, que corresponderá con el identificador de usuario. El resto del guardado se realizará como un guardado normal.

La fase de pruebas consistió en realizar unas cuantas vistas previas y comprobar que todo funcionaba bien. Hicimos especial hincapié en que el archivo temporal se sobre-escribiera y no se creara uno nuevo cada vez que hacíamos un nueva vista previa.

4.6 Iteración 6. Editor de código.

Para completar esta iteración hemos añadido un panel que se alternará con el del lienzo, y un botón que ocultará uno u otro de forma alternada. Este panel lo podemos ver en la Ilustración 11 de la sección 2.2 (Requisitos). Debido a la complejidad que supone la implementación de un editor de código desde cero, buscamos por internet alguna forma de facilitarnos la tarea, y lo que encontramos fue AceEditor.

AceEditor es un editor de código integrable escrito en Javacript. Consigue las funcionalidades y rendimiento de los editores nativos como Sublime, Vim y TextMate. Puede ser fácilmente embebido en cualquier página y aplicación Javascript. Está realizado bajo licencia BSD, lo que nos es de gran ayuda ya que permite el uso de su código de forma completamente libre,

siempre y cuando indiquemos el copyright, incluso aunque nuestra aplicación vaya a ser software privativo.

Desarrollo: La inclusión de AceEditor en cualquier aplicación es muy sencilla basta con seguir una serie de pasos muy simples:

- Introducimos un código HTML en el panel de código que contendrá un vacío con un identificador = "editor". Además tendrá como referencias, dos ficheros Javascript: ace.js (que podremos encontrar en el código fuente de ace-editor) y cargar_editor.js (que estará en la misma carpeta que el código fuente del ace-editor).
- En el fichero cargar_editor.js tendremos una función llamada Refrescar, la cual será llamada cada vez que se muestre el panel de código. Esta función hará lo siguiente:

```
var caja_texto = document.getElementById("editor");      // accedemos //   al HTML
// creamos el editor
var editor = ace.edit(document.getElementById("editor"));
// accedemos al código   del sinóptico que estamos editando
var value = window.parent.lienzo.getSvgString();
editor.setValue(value); // insertamos el código en el editor
// aplicamos un tema y un modo al editor
editor.setTheme("ace/theme/xcode");
editor.getSession().setMode("ace/mode/svg");
```

A parte de esto también añadimos dos funciones nuevas a editor.js:

- **Vista_codigo**: Ocultamos el panel del lienzo y mostramos el de código. También tenemos que actualizar su contenido para que se vean reflejados los cambios recientes en el dibujo. Esto último lo hará la función *Refrescar*.
  Vista_lienzo: Accedemos al código SVG mediante la función proporcionada por el aceEditor: *getValue()*. Actualizamos el contenido del lienzo mediante la función *loadFromString* de la clase Editor.

Las pruebas consistieron alternar ambas vistas, lienzo y código, cambiando cosas en una y otra, y comprobando que los cambios realizados en una se veían reflejados en la otra y viceversa. Al principio tuvimos algún que otro problema con el acceso entre la clase Lienzo y el AceEditor, ya que estaban en niveles de la jerarquía de carpetas distintos, pero no nos costó mucho solucionarlos.

4.7 Iteración 7. Nuevo componente dinámico: Switch.

Una vez completadas todas las funcionalidades básicas y con el editor prácticamente funcionando, analizamos las peticiones de nuevas funcionalidades que nos habían hecho los usuarios que estaban probando la herramienta y decidimos incluir un nuevo componente.
El nuevo componente consistía en un condicional mejorado, en lugar de alternar entre dos imágenes diferentes, lo haría entre dos o más.

Desarrollo:

Diseño

Al igual que con el resto de los componentes, lo primero que hacemos es crear el código SVG base del componente. Tendrá como base el condicional pero tendrá en primera instancia un solo item. Contará con los siguientes atributos:

- PID: Identificador del proceso al que representa.
- Visible: Identificador del ítem que está visible en el momento actual.
- Default: Item que se mostrará si el valor recibido no encaja entre los intervalos de valores de ningún ítem.

A su vez cada ítem tendrá sus propios atributos:

- valorMaximo: Rango superior de los valores entre los que se mostrará el ítem poseedor del atributo.
- valorMinimo: Rango inferior de los valores entre los que se mostrará el ítem poseedor del atributo.

Tras esto podemos pasar a implementar la funcionalidad de la asignación de valores a los atributos. Para ello contamos con una ventana, como la de la Ilustración 29, donde iremos introduciendo datos.

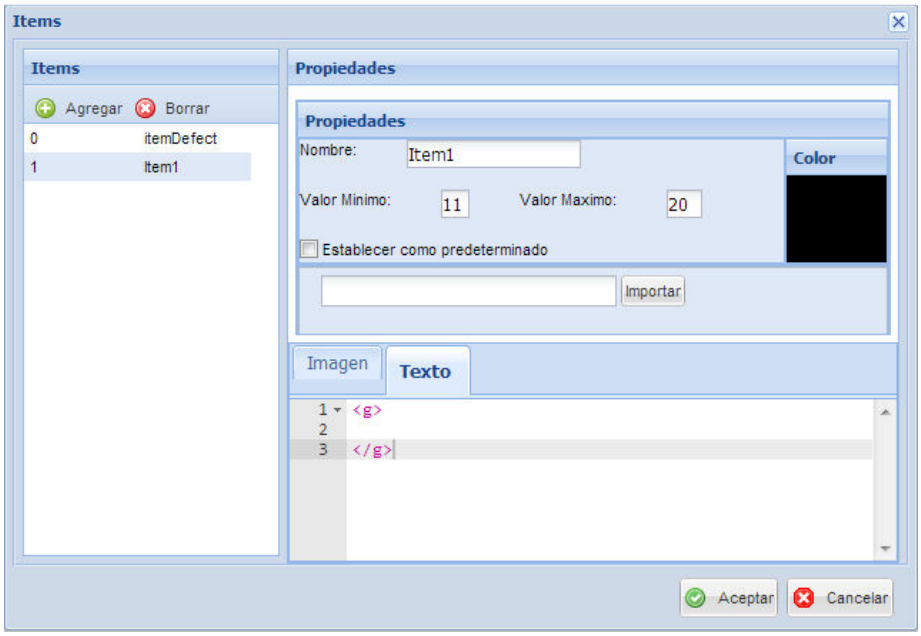

Ilustración 29 - Ventana creación switch

Podemos dividir la ventana en tres paneles:

1. Panel Izquierdo: Sirve para seleccionar el ítem que queremos editar, además nos da la posibilidad de añadir nuevos ítems o borrar los que ya tenemos. Está implementado a través de una lista asociada a un almacén de datos que guardará, no solo los identificadores de los ítems, sino también el resto de sus atributos.

2. Panel Central-Superior: Permite la edición de atributos, nombre valor mínimo, valor máximo y un botón de chequeo para asignar el ítem que se mostrará por defecto.

3. Panel Central-Inferior: Nos permite editar el código SVG del ítem. Podemos copiar el código de nuestros diseños copiando el contenido de las etiquetas <svg> y pegándolo en este mini-editor Ace.

Funcionalidad:

Para la creación del switch se cuenta con las siguientes funciones:

- mostrar_ventana_switch: Limpia los registros de la ventana, además de reiniciar el mini-editor, y la muestra.
- Cargar_datos_switch: Recarga el almacén con los datos del switch antes de mostrarla. Borramos el almacén, guardamos el id del ítem por defecto y empezamos a recorrer los ítems. Por cada ítem creo un registro en el almacén con sus respectivos atributos de ítem. Refresco el listado del Panel Izquierdo.
- btnAgregarItem_Click: Añado un nuevo registro al almacén, pero sin valores. Refresco el listado.
- Guardar_cambios_item: Selecciono otro ítem actualizo los datos del que tenía previamente seleccionado con los valores que están en los registros del Panel Central-Superior.
- listItems_Click: Cargamos los valores de los paneles centrales con los valores del ítem seleccionado sacados del almacén.
- btnAceptarSwitch_Click: Guardamos los datos del ítem actualmente seleccionado, llamando a la función *guardar_cambios_item*. Recorro el almacén recreando el switch para finalmente agregarlo al código SVG que tenemos en el lienzo.

Las pruebas de este componente fueron algo más complicadas que las de los demás, obviamente por la mayor complejidad del mismo.

Podemos dividir los test en dos partes:

1. Test para el selector de ítems:

Para comprobar que nuestra ventana de selección de ítems funciona como se espera se han realizado algunas pruebas de caja blanca. Para cada una se ha realizado un diagrama de flujo y con el depurador se ha comprobado que las funciones tenían el comportamiento esperado. Las pruebas han permitido comprobar cómo se añade, se borra, se selecciona y se valida sido un switch. Como ejemplo mostramos los procesos seguidos para agregar y borrar un item:

Prueba 1. Agregar ítem: Como podemos ver en la Ilustración 30 (a), existe una bifurcación a la hora de crear un nuevo ítem. En caso de no haber ítems en el almacén crea uno nuevo con valorMinimo = 10, valorMaximo = 20 y predeterminado = true.

Prueba 2. Borrar ítem: En esta función lo importante es que si el ítem borrado era el ítem predeterminado, se ponga como predeterminado el primer ítem del listado. Podemos ver esta

comprobación en el diagrama de flujo de la Ilustración 30 (b). Siempre debe haber un ítem predeterminado en el switch.

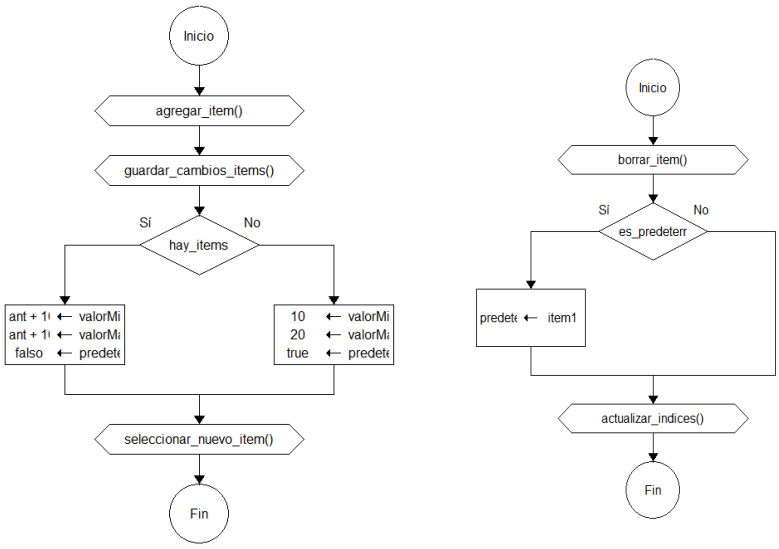

Ilustración 30 – Ejemplo de pruebas del switch

En el proceso de creación final del switch, se comprueba que las últimas modificaciones en el Panel Central Superior se almacenan correctamente en el almacén, que los hijos del switch que está en el lienzo han sido borrados y que todos los nuevos ítems y los nuevos parámetros son correctamente introducidos en el nuevo switch.

Tras comprobar que los resultados eran los esperados pasamos a verificar la funcionalidad del componente.

2. Test para funcionalidad switch:

Para probar su funcionalidad hacemos varias pruebas de caja negra asignando valores diferentes a los ítems en cada prueba. También hacemos pruebas para diferentes números de ítems.

Prueba 1. 2 items, exclusión, rango incompleto

Item1 : 0 – 100

Item2 : 150 – 1000 predeterminado

Esta prueba tuvo un resultado satisfactorio. El componente mantuvo el comportamiento esperado.

Prueba 2. 2 items, inclusión, rango incompleto

Item1 : 0 – 150 predeterminado

Item2 : 100 – 1000

- Podemos ver los resultados de la prueba en la Tabla 3:

Valor	Resultado	Correcto
111	Item1	SI
37	Item1	SI
196	Item2	SI
22	Item2	NO
69	Item2	NO
117	Item2	SI

Tabla 3 - Resultados prueba funcionalidad switch

Después de mostrar el Item2, éste se muestra siempre. Esto no debería ser así. El error se resolvió guardando en una variable el elemento que estaba actualmente visible, de no ser así ese elemento nunca se ocultaba.

Después de esta serie de pruebas dimos el visto bueno al switch y publicamos los nuevos cambios en el servidor de desarrollo. Nuestro cliente quedó muy satisfecho.

Tras estas siete iteraciones pudimos entregar a los clientes una aplicación muy funcional que se adecuaba a los que en primera instancia se especificó en los requisitos. Aun así en este punto los clientes seguían encontrando pequeños bugs, que intentábamos solucionar lo más rápidamente posible, y continuaban solicitando nuevas funcionalidades para adecuar la aplicación a sus necesidades.

5. Evaluación y pruebas

Una vez completada la fase uno de la aplicación, fase durante la cual he desarrollado mi proyecto, se pasó a una fase general de pruebas en la cual gente del equipo de diseño y algunas personas del equipo de CIC ajenas al proyecto fueron probando la aplicación y buscando bugs.

Para realizar las pruebas necesarias se desplegó la aplicación en el servidor de desarrollo de CIC, al igual que la fase beta que desplegamos tras la iteración 4. La aplicación de IDbox en este servidor cuenta con un usuario de prueba, que es el que se usará para los tests, que tiene permisos para hacer uso de todas las funcionalidades excepto la de guardar en base de datos.

Tras esto nos centramos en corregir los errores que ocurrían con mayor frecuencia dejando el resto para más adelante, sobre todo si no eran errores críticos que supusieran el incorrecto funcionamiento de la aplicación. A día de hoy la mayoría han sido corregidos pero la aplicación continúa en desarrollo y siguen detectándose fallos, los cuales están en fase de ser subsanados.

Queda decir que encontramos nuevos errores cada día debido al constante incremento de funcionalidades que nos vemos obligados a añadir a petición de nuestros clientes. Por lo que esta fase de evaluación y pruebas aún continúa actualmente.

6. Conclusiones y trabajos futuros

6.1 Conclusiones

Cuando me encargaron el proyecto, lo primero que pensé fue que la creación de un editor de gráficas sería una tarea sencilla, que no entraba dentro de la complejidad que yo entendía para un proyecto de fin de carrera. Al principio pensé, viendo la cantidad de editores que hay por la web, que reproducir algo parecido e incrustarlo en la aplicación IDbox sería coser y cantar. Estaba muy equivocado.

Es a la hora de añadir a algo tan complejo como es IDbox, una nueva funcionalidad respetando lo que ya está hecho, donde encontré la mayor parte de los problemas. IDbox es un sistema con una estructura muy marcada, sigue un patrón de diseño concreto (MVC) y esto es algo que hay que respetar. Tuve que crear mi propia estructura MVC y realizar pequeños protocolos estándares a la hora de acceder a algunas de las funcionalidades de IDbox ya implementadas. Aunque ya conocía lo que era MVC en teoría, nunca había trabajado realmente siguiendo este patrón de diseño, por lo que fue algo duro al principio hasta que conseguí acostumbrarme.

Otro aspecto al que he tenido que acostumbrarme es a trabajar en un equipo de desarrolladores. Ya había realizado trabajos en grupo durante la carrera, sin embargo trabajar en una aplicación tan grande, donde varias personas van construyendo partes de la aplicación y tener que trabajar con un software de control de versiones (subversión) para unificar todo el trabajo sin que se creen conflictos en el código, era algo completamente nuevo para mí. A lo que, por supuesto, acabé acostumbrándome.

Dicho todo esto debo decir que mi experiencia ha sido muy positiva. Me he divertido desarrollando esta aplicación y he aprendido mucho de lo que es trabajar en una empresa de desarrollo software.

6.2 Trabajos futuros

Actualmente sigo en CIC continuando con la fase 2 del desarrollo del diseñador de sinópticos. Como dije al final del apartado Evaluación y Pruebas, los clientes nos siguen pidiendo nuevas funcionalidades. Ahora mismo estamos trabajando en nuevos componentes como una barra de progreso circular, un botón para mandar comandos, tooltips al pasar por los objetos, etc… Y también implementando la posibilidad de añadir animaciones a los componentes actuales, como parpadeos o animaciones de movimiento.

7. Bibliografía

Todas las referencias web han sido consultadas durante los meses de mayo a junio de 2013

[1] Características CorelDRAW: http://www.corel.com/corel/product/index.jsp?pid=prod4260069&cid=catalog3440075&segid=7000013&storeKey=es&languageCode=es

[2] Características Serif DrawPLus: http://www.serif.com/drawplus/

[3] Características Flatpaint: http://www.fatpaint.com/

[4] Características Macromedia FreeHand: http://www.adobe.com/products/freehand/

[5] Características Adobe Illustrator: http://www.adobe.com/products/illustrator/features.html

[6] Características Inkscape: http://inkscape.org/doc/index.php?lang=es

[7] Características OpenOffice Draw: http://www.openoffice.org/product/draw.html

[8] Características PhotoLine: http://www.pl32.com/

[9] Características sk1: http://sk1project.org/modules.php?name=Products&product=sk1

[10] Características svg-edit: https://code.google.com/p/svg-edit/

[11] Características Xara Xtreme: http://www.xara.com/eu/

[12] Documentación del producto IDbox: http://idbox.cic.es/descargas/documento-es.pdf

[13] Tutoriales Microsoft (MVC): http://msdn.microsoft.com/es-es/library/gg416514(v=vs.98).aspx http://msdn.microsoft.com/es-es/library/dd381412(v=vs.98).aspx

[14] J.David Eisenberg. “SVG Essentials: Producing Scalable Vector Graphics with XML”. O'Reilly Media, Inc., 2002

[15] Andrew Troelsen. “Pro C# 5.0 and the .NET 4.5 Framework”. Apress; Edición: 6 (2012)

[16] Información ext .NET: http://www.ext.net/

[17] Roger Pressman. “Ingeniería del Software: Un enfoque práctico”. McGraw-Hill/Interamericana de Espana, S.A., 2007

[18] Metodologías software: Apuntes de Ingeniería de Software 2, Rafael Duque (2012).

[18] Matthew Eernisse. “Ajax web applications”. SitePoint Pty Ltd (2006)

[19] Jeremy Keith. “DOM Scripting: Web Design with JavaScript and the Document Object Model”. friendsofED (2005)

[20] Aquilino Rodríguez Penin. “Sistemas SCADA”. (2ª edición) Marcombo, S.A (2007)

[21] Microsoft Visual Studio: http://www.microsoft.com/visualstudio/esn/2013-preview