Proyecto de Fin de Carrera

Pizarra Web Compartida

Albert Llop

6 de octubre de 2008

Índice general

Capítulo 1

Trabajo Previo

1.1. Especificación

Este proyecto surge de la necesidad de realizar presentaciones a distancia, mayoritariamente por videoconferencia. Las videoconferencias están en el día a día de las empresas, y por supuesto, las presentaciones en powerpoint están en la vida diaria de cualquiera. El realizar estas presentaciones a distancia crea una serie de problemas:

- Cada *usuario* ve el documento individualmente, por lo cual no hay una sincronía entre lo que todos ven. Cada uno puede estar mirando una hoja distinta, o a un punto distinto.
- A diferencia de en una presentación real, el poniente carece de una pizarra o una superficie donde hacer explicaciones cuando las diapositivas no son suficiente.

Estas dos carencias básicas hacen que se pierda un gran porcentaje del tiempo de dichas conferencias en forzar esa sincronía, con explicaciones constantes de en qué diapositiva se está, o a qué punto mirar, o en simular dicha pizarra, con explicaciones verbales en vez de gráficas.

Este proyecto por tanto intenta ser una solución para dichas empresas (o cualquier usuario), de forma que se pueda disponer de ambas funcionalidades de forma sencilla y accesible. Se pretende explorar el mundo del desarrollo de aplicaciones web interactivas, profundizando en la comprensión de las capacidades de las tecnologías actuales, disponibles al público en los navegadores típicos.

1.1.1. Limitaciones

Considerando que el mayor uso de este software va a ser por parte de empresas u otras organizaciones, más que particulares, hace que se tengan que considerar una serie de limitaciones. Éstas son las básicas:

- Las empresas generalmente no permiten instalar nuevo software en sus ordenadores.
- Hay que tener en cuenta que muchas empresas tienen instalados firewalls muy restrictivos.
- Los documentos no tienen porqué ser vistos por todo el mundo, debe de haber algún tipo de seguridad que permita que solo la gente adecuada pueda estar en la presentación.

1.1.2. Funcionalidades

Una vez definida los objetivos básicos que queremos alcanzar, y las restricciones, se pueden empezar a formalizar las funcionalidades. La solución más simple que cumpla las limitaciones, y que permita alcanzar dichas funcionalidades básicas, es la realización de una **web** que permita cargar documentos y dibujar sobre ellos de forma compartida con otros usuarios. Cualquier persona puede disponer de un explorador web, y generalmente funcionará a través del firewall. El tercer requisito se puede cumplir fácilemte, pues la seguridad web es un campo suficientemente desarrolado para ello. A continuación se numeran más detalladamente las funcionalidades que esta web ha de tener, separados en funcionalidades del sistema y de la pizarra en si.

Funcionalidades del Sistema

De entre las múltiples posibilidades a la hora de plantear el funcionamiento de la web, se ha considerado interesante estructurar la web como una "comunidad" en que, simplificando al máximo, los usuarios se registran, pueden subir sus documentos, e invitan a otros usuarios a que se unan a sus presentaciones. Por lo tanto:

Gestión de Usuario Se tiene que poder registrar, hacer login y salir. Todas las opciones tienen que ser modificables por el usuario, por ejemplo, la contraseña.

Gestión de Grupos (Opcional) Poder crear grupos, invitar a usuarios a dichos grupos.

Gestión de Pizarras Cada usuario tiene que poder crear sus pizarras (el concepto de pizarra y sus funcionalidades se detallan en la sección siguiente), editarlas, y eliminarlas. Tiene que poder invitar a otros usuarios y/o grupos a partizipar en esa pizarra.

Participación en otras Pizarras Cada usuario tiene que poder ver las pizarras a las que ha sido invitado, acceder y salir de ellas. (Opcional) Rechazar y solicitar invitaciones.

Funcionalidades de la Pizarra

Se considera una pizarra como un lugar donde poder escribir, dibujar, al cual se le ha cargado unas imágenes de fondo, que se podrían considerar las diapositivas de una presentación. Una persona que tenga que hacer dos presentaciones a partir de dos archivos distintos, tendrá que crear dos pizarras distintas, e invitar a la gente adecuada. Se puede invitar a gente distinta en cada pizarra que se haya creado. La figura 1.1 es una representación básica de cómo se vería una pizarra en funcionamiento. Arriba habrían las herramientas de dibujo, y a la derecha la lista de usuarios conectados. Las funcionalidades deseadas para las pizarras son las siguientes:

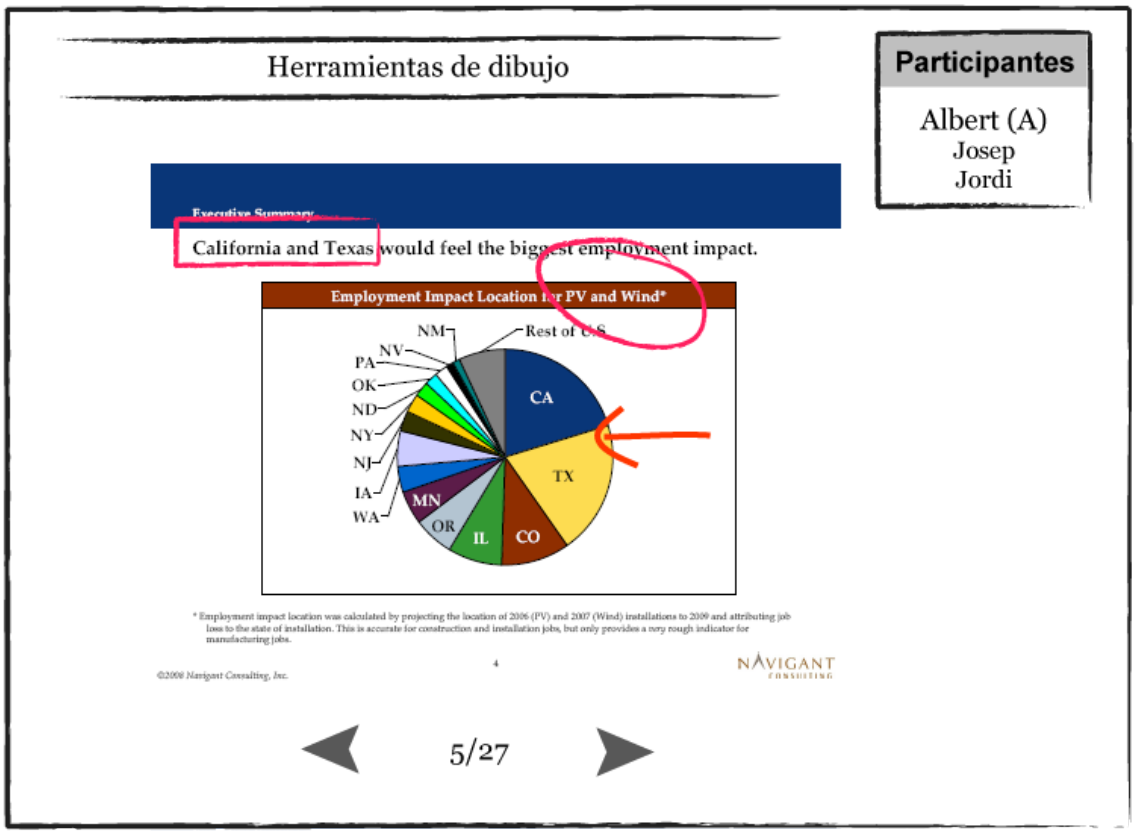

Figura 1.1: Concepto básico de una pizarra

Carga de documento Cada pizarra tiene la posibilidad de añadir un documento que se cargará como imágenes de fondo. Dichas imágenes se pueden cargar de múltiples maneras.

- Una por una en formato de imagen JPG/PNG.
- Un zip/rar con el conjunto de imágenes.
- (Opcional) Mediante un PDF.
- (Opcional) Mediante una presentación PowerPoint directamente.

Multipágina Cada imagen tiene que ser una página diferente. Se tiene que poder avanzar y retroceder entre ellas.

Puntero Se tiene que poder ver el puntero del "administrador", de forma que los usuarios pueden ver qué está señalando en ese momento, en vez de esperar a que dibuje un círculo, por ejemplo.

Herramientas de dibujo Existen múltiples posibilidades para esto, y se intentarán implementar el máximo número posible, pero se establecen unos mínimos:

Lápiz Con selección de color y grosor.

Lineas rectas Con selección de color y grosor.

Texto Con selección de color, fuente y tamaño. (Estos dos últimos opcionales)

Goma de borrar Para poder eliminar objetos con la mayor sencillez posible.

Subrallador (Opcional) Para todas las herramientas, poder seleccionar modo de subrallado, en que se harán los dibujos semitransparentes a modo de subrallador, en vez de sólidos.

Cajas y elipses (Opcional)

Imágenes (Opcional) Se desea la posibilidad de añadir imágenes extra, a parte de la del documento de fondo.

Guardar Se tiene que poder guardar la situación actual de la pizarra para cargarla posteriormente.

Formato en capas (Opcional) Ya se entiende que cada pizarra tendrá dos capas, la de la imagen de fondo, y en la que se hagan todos los dibujos. Sería interesante hacer que se pueda dibujar por capas, crearlas, eliminarlas, moverlas arriba/abajo.

Exportar Se tiene que poder exportar la situación de la pizarra en algun formato (pdf, imagen) y/o poder imprimirse.

Lista de usuarios Listar los usuarios que están en ese momento conectados en la pizarra.

Chat (Opcional) Posibilidad de establecer un chat entre los usuarios conectados. (Opcional) Poder guardar dicho chat junto con el estado de la pizarra.

Para ayudar a entender el concepto de lo que será el sistema, la figura 1.2 representa un diagrama de clases extremadamente simplificado.

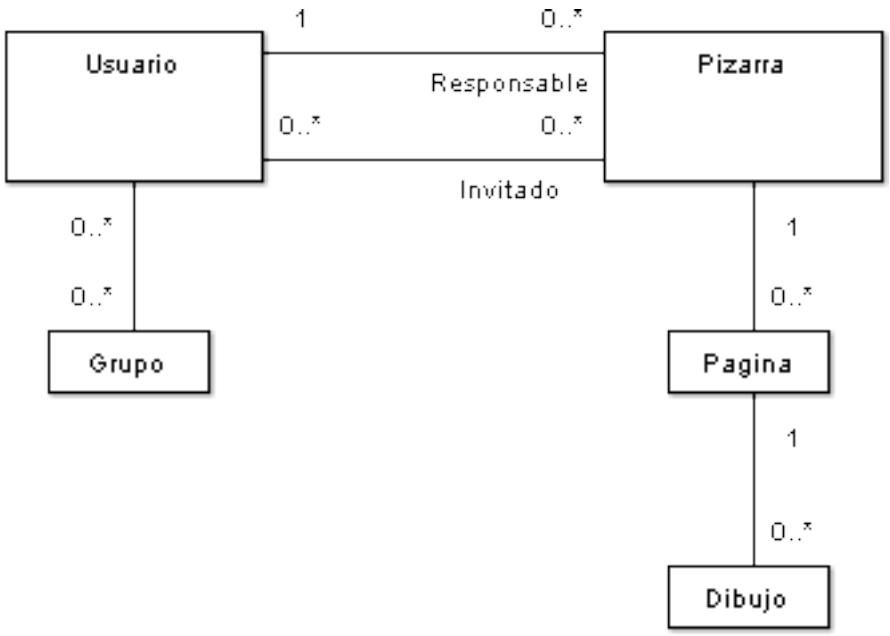

Figura 1.2: Diagrama de clases básico

1.1.3. Tecnología

Una explicación más detallada de la tecnología que se va a utilizar se podrá encontrar en las próximas secciones, pero hay una serie de requisitos que se afectan a la tecnología que se use, que deben ser considerados.

Servidor El servidor no tiene que requerir más de lo que sería un servidor web típico. Apache/MySQL sobre linux. Otros requisitos se admiten, pero tienen que estar accesibles de forma sencilla (binarios en forma de paquete, por ejemplo) para las distribuciones típicas de Linux. Nada de compilar fuentes, por ejemplo.

Cliente Los navegadores más importantes tienen que estar soportados, Internet Explorer, Firefox y Safari como mínimo. Se intentará alcanzar el mayor grado de compatibilidad con Explorer 6, y en el peor caso, que cumpla las funcionalidades básicas.

Software Puesto que los usuarios no tienen que tener que instalar ningún software nuevo, es altamente deseable que la web sea lo más sencilla posible. Se hará todo lo posible por evitar utilizar tecnologías que el usuario no tiene porqué tener instalados en su ordenador. Se va a intentar cumplir el requisito de que el usuario solo tiene que tener funcionando un navegador web moderno. En cuanto a la programación del esqueleto de la web, se considera interesante utilizar un lenguaje dinámico más moderno para agilizar el proceso, en vez del clásico PHP o ASP. También, el interés propio de explorar nuevas áreas, puesto que ya se tiene amplia experiencia con PHP, Ruby es un lenguaje muy interesante, y será considerado en la etapa de diseño.

1.2. Tecnología

Se ha decidido realizar una web con una serie de funcionalidades, pero incluso en ese entorno existen múltiples lenguajes y plataformas para todo tipo de desarrollos. Este documento pretende considerar las posibilidades existentes, y escoger la más adecuada para este proyecto a partir de una serie de criterios.

En un primer acercamiento se pueden diferenciar dos partes claras que forman este proyecto. Primero, el desarrollo de la pizarra, implementando el máximo de funcionalidades establecidas en la especificación, y segundo, el desarrollo del sistema, con su gestión de usuarios, grupos y pizarras.

1.2.1. Pizarra

A la hora de considerar las posibles tecnologías que permitan implementar una pizarra de este tipo, aparecen múltiples opciones que, en mayor o menor medida, podrían funcionar. Sin embargo, en la especificación se ha definido que la solución debe ser accesible desde una página web, a través de firewalls rigurosos, y con el mínimo de *software extra* posible. Ésto nos deja básicamente con tres posibles opciones, descartando muchos de los lenguajes más clásicos de programación, como podría ser C++.

Dichas opciones son `JavaScript`, `Flash` o `Java`. Otro tipo de aplicaciones como podría ser `ActiveX` queda descartado desde el principio por su dificultad de ejecutar en exploradores que no sean Internet Explorer, y por estar aún más limitados en entornos restrictivos como son los ordenadores de las empresas. Los tres considerados son suficientemente conocidos, y están soportados por la gran mayoría de los ordenadores de hoy día.

Para poder hacer una comparativa objetiva entre ellos, los criterios a seguir serán los siguientes:

Facilidad de implementación Una tecnología que permita desarrollar de forma más rápida también permite desarrollar más y con menos errores.

Calidad del resultado Es muy posible que con alguna tecnología el resultado que se llegue a obtener sea de mejor calidad por múltiples razones.

¿Accesible por todo el mundo? ¿Es posible que algún usuario tenga alguna limitación que no le permita usar el programa debido al uso de esta tecnología?

Otras motivaciones Se considerará también muy importante características como el ser una tecnología novedosa, algo con el que el autor aún no haya trabajado, tendencias de mercado, etc.

Viabilidad Pueden haber razones por las cuales una tecnología es simplemente inviable, ya sean monetarias, de complejidad, adquisición de licencias, etc.

Flash

Adobe Flash (anteriormente Macromedia Flash) era originariamente un entorno enfocado al desarrollo de animaciones vectoriales, pero su gran popularidad ha hecho que se haya ido expandiendo hasta convertirse en una plataforma multimedia mucho más interactiva, gracias a `ActionScript`, el lenguaje mediante el cual se puede programar todo tipo de aplicaciones en flash.

Facilidad de implementación Es muy posible que de las tres tecnologías esta fuera la que permita una implementación más sencilla de la pizarra en si, pues Flash proporciona todas las herramientas necesarias para tratar los dibujos que se necesitan hacer. Un magnífico ejemplo de pizarra interactiva puede encontrarse en `www.imaginationcubed.com`. En cuanto a las funcionalidades de pizarra compartida, Flash ofrece *Flash Media Server*[1], que soluciona todas las necesidades de aplicaciones en tiempo real y compartición de objetos.

Calidad del resultado Los resultados que se pueden obtener con flash en una aplicación interactiva de este tipo pueden ser más que satisfactorios. La calidad gráfica, así como la interactividad serán tan buenas como el programador/diseñador sea capaz.

¿Accesible por todo el mundo? Flash es una tecnología muy extendida por todo el mundo, pero por desgracia suele utilizarse para aplicaciones que muchas empresas podrían considerar como inadecuadas en un espacio de trabajo (juegos en flash o youtube, por ejemplo). Si bien es cierto que la posibilidad de que un ordenador moderno no disponga de flash es muy reducida, no es algo imposible, y mucho menos en el contexto de las empresas.

Otras motivaciones Flash es una tecnología que lleva muchos años en el mercado y que ha madurado considerablemente. Sin embargo suele requerir de capacidades de diseñador/animador más que de programador, por tanto no se considera muy adecuada para el perfil del autor.

Viabilidad El mayor problema de Adobe Flash es el hecho de que es una tecnología cerrada. La licencia de Flash CS3 Profesional, necesario para desarrollar el programa, cuesta \$699[2], y la licencia para el servidor Flash Media Server asciende a \$4500[3]. Ambos productos están fuera del alcance en cualquier implementación realista. Si bien es cierto que el servidor de desarrollo es gratis, se considera importante que sea posible utilizar el software una vez acabado, si no por cualquier empresa, al menos por la mayoría.

Java

Java es un lenguaje de programación en toda regla, que ofrece la posibilidad de generar los llamados applets, enfocados al entorno web, con todas las ventajas (e inconvenientes) de un lenguaje de programación normal y corriente. Dichos applets son integrables en las webs sin problema, siempre y cuando el usuario tenga instalado la JVM (*Java Virtual Machine*).

Facilidad de implementación Java permite hacer prácticamente cualquier cosa, el único problema es que no es un lenguaje pensado especialmente para aplicaciones interactivas como en este caso. Además, para que dichos applets interactúen con el sistema, y por ejemplo, se puedan guardar pizarras, sería conveniente que se funcionase con Servlets y JSP's, lo cual no lo hace más difícil, pero si más restrictivo.

Calidad del resultado Al no ser un lenguaje pensado para este tipo de aplicaciones, es posible que el resultado no sea de la calidad que se espera, muy pesado y engorroso para el usuario por tener que cargar el applet. Java suele utilizarse para programas más grandes, o que necesitan de mayor procesado que lo necesario por una pizarra interactiva.

[1] Adobe, Flash Media Server Products - `http://www.adobe.com/products/flashmediaserver/`

[2] Adobe, Flash CS3 Professional - `http://www.adobe.com/products/flash/?ogn=EN_US-gntray_prod_flash_home`

[3] Adobe, Flash Media Interactive Server 3 - `http://www.adobe.com/products/flashmediainteractive/`

¿Accesible por todo el mundo? De forma similar al flash, java está instalado en la mayoría de ordenadores personales, aunque es posible que en menor medida.

Otras motivaciones El autor ya ha realizado numerosas aplicaciones en java, incluyendo applets, así como servlets y JSP's, con lo cual se considera poco interesante repetir la experiencia.

Viabilidad A pesar de lo que parecen ser numerosos problemas, Java sería una opcion perfectamente factible, dentro de las capacidades del autor, y que resultaría en un programa quizá no tan agradable visualmente como si funcional. Se puede desarrollar en java de forma libre, no habría ningún gasto extra, ni problema de licencias.

JavaScript

JavaScript, a diferencia de las otras opciones, es un lenguaje interpretado por los navegadores que permite la ejecución de, en principio, pequeñas acciones dentro de la página web. Dichas acciones pueden ser desde hacer pequeñas operaciones con datos introducidos en un campo de texto, habilitar o desabilitar elementos web, hasta otras cosas mucho más complejas. Javascript es en realidad un lenguaje muy completo que permite hacer una gran variedad de cosas, y esto se ha ido demostrando conforme han ido pasando los años. Ejemplos de webs con un gran uso de javascript podrían ser, por ejemplo, google docs[4] o google mail[5].

Facilidad de implementación Por desgracia, Javascript se ha utilizado mayoritariamente para pequeñas operaciones como las descritas anteriormente, lo cual ha hecho que haya, en general, poca experiencia a la hora de desarrollar aplicaciones más complejas con él, y lo que ello conlleva (falta de documentación, ejemplos, librerías, etc). Otro problema bastante grande, es el hecho de que sea un lenguaje interpretado por el explorador, y que no todos los exploradores tengan una implementación similar de Javascript. A pesar de los esfuerzos del W3C[6] aún hay diferencias entre implementaciones. Ésto hace que se tengan que tener en cuenta más factores a la hora de programar, haciendo el proceso más complicado.

Calidad del resultado Javascript permite crear aplicaciones muy ágiles para el usuario, utilizando técnicas ya extendidas como Ajax (Asynchronous JavaScript And XML), creando una experiencia de usuario muy positiva. Si bien no está pensado para realizar aplicaciones gráficas, existen ejemplos funcionales de una pizarra similar a la que se quiere implementar, usando Javascript. Existen librerías que permiten trabajar con formas sencillas, como podría ser jsgraphics[7], aunque se intentará encontrar alguna solución mejor.

¿Accesible por todo el mundo? Ésta es la tecnología que más personas podrán disfrutar. Solamente hace falta tener un navegador relativamente moderno (Internet Explorer 6+, Mozilla Firefox, Safari, y muchos otros), sin ningún tipo de añadido. El problema reside en el programador, por crear código que sea entendible por todos, no en el usuario.

Otras motivaciones Javascript está en auge en estos momentos, se está por fin dando un uso completo a todo su potencial por numerosas compañías, y los resultados son más que sorprendentes. La agilidad de las aplicaciones hace que se empiece a utilizar la web para

[4] Google Docs - `http://docs.google.com`

[5] Google Mail - `http://mail.google.com`

[6] World Wide Web Consortium - `http://www.w3c.org`

[7] High Performance JavaScript Vector Graphics Library - `http://www.walterzorn.com/jsgraphics/jsgraphics_e.htm`

múltiples tareas que antes estaban relegadas solamente a programas individuales. Existen implementaciones de todo tipo de aplicaciones con javascript, desde clientes de mensajería instantánea, a videojuegos, pasando por clientes de correo, procesadores de texto u hojas de cálculo. El autor aún no ha trabajado con este lenguaje más que en contadas ocasiones y de forma mínima, haciéndolo muy atractivo a la hora de desarrollar un proyecto como este.

Viabilidad No hay ninguna razón que haga el uso de javascript inviable, existen pizarras compartidas online ya implementadas con gran éxito, pero no cumplen los requisitos que este proyecto se ha planteado, o son cerradas/de pago.

Conclusión

Después de considerar los criterios que se han establecido, se ha decidido utilizar `Javascript` como técnica para implementar la Pizarra. Flash hace bastante inviable su desarrollo por el alto costo de sus licencias, y aunque se puedan usar licencias de prueba, el producto final sería inviable de utilizar por cualquier empresa que no disponga ya de las licencias de Flash Media Server, y por supuesto totalmente imposible en caso de querer ponerla en funcionamiento por parte del autor. En cuanto a Java, es una solución interesante, y que funcionaría sin demasiados problemas, pero se pretende que sea algo ágil, que cualquiera pueda utilizarlo, y Java suele tener problemas con ello. Javascript es interesante en todos los aspectos, y se considera que permitirá realizar una implementación excelente, además de servir para profundizar más los conocimientos del autor en la materia web actual.

1.2.2. Sistema

Hasta ahora se ha considerado solamente cómo implementar las Pizarra, que si bien es el elemento más importante del proyecto, no es el único. Las funcionalidades que se han definido requieren de otro tipo de lenguajes, de los cuales hay una gran variedad, y se considerarán a continuación bajo los siguientes criterios, similares al apartado anterior.

Facilidad de implementación

Calidad del resultado

Disponibilidad en servidores comunes

Otras motivaciones

Viabilidad

Las opciones a considerar serán `PHP`, `JSP/Servlets Java` y `Ruby`. Existen múltiples lenguajes que permiten programar webs dinámicas, como podrían ser python, perl, o incluso C. Sin embargo no son tan populares en cuanto a desarrollo web se refiere, por lo tanto solamente se considerarán las que son mayoritarias actualmente.

PHP

PHP se describe a si mismo como[8]:

[8] `http://www.php.net`

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

PHP se ha convertido en el lenguaje de scripting más popular en la actualidad, utilizado en más de 20 millones de sitios web[9]. Ésto, añadido a su licencia libre, hace que exista una comunidad enorme, con todos los beneficios que ello conlleva.

PHP permite generar contenido dinámicamente en el servidor, dependiendo de las numerosas variables existentes. Se pueden "incrustar" trozos de código PHP en el html de forma que PHP genere ese contenido haciendo las operaciones necesarias, como por ejemplo, consultando una base de datos.

Facilidad de implementación PHP se considera un lenguaje simple que permite desarrollar sitios de tamaño pequeño o medio con relativa simplicidad. Existen múltiples scripts ya programados que incluso podrían llegar a ayudar en el proceso de implementación del sistema.

Calidad del resultado El amplio uso de PHP en sitios web hace que sea un sistema completamente depurado con mínimos agujeros de seguridad, lo cual ayudado por la gran cantidad de documentación de que se dispone, hace que las webs resultantes no tengan nada que envidiar a las producidas por otros lenguajes. Realmente el límite está en manos del programador, no del lenguaje.

Disponibilidad en servidores comunes PHP viene con cualquier distribución de linux hoy día, y se puede instalar en prácticamente cualquier sistema operativo.

Otras motivaciones El autor de este proyecto ya tiene amplia experiencia en la programación de webs con PHP. No aportaría ningún conocimiento nuevo, más que incrementar la experiencia ya existente.

Viabilidad Este proyecto en PHP sería totalmente viable.

JSP/Servlets Java

JSP (JavaServer Pages) se puede considerar como una abstración de los Servlets, en un lenguaje más alto. Un JSP se compila generando un servlet, que es código Java, el cual se vuelve a compilar con un compilador Java tradicional.

Simplificando, JSP funciona de forma muy similar a PHP, pero con todo el soporte de Java, y lo que ello conlleva. Hace que sea posible utilizar cualquier clase Java para generar contenidos dinámicos, de la misma forma que PHP usa sus librerías.

Facilidad de implementación JSP es tan simple como Java, y tiene todas las características necesarias para hacer cualquier sitio web. Sin embargo, Java no está tan enfocado a páginas webs como podría ser PHP. Se tiene una gran ventaja en el hecho de que se puede usar cualquier código Java, por tanto, cualquier librería ya hecha, pero también es cierto que al no ser enfocado principalmente a páginas web, pueda hacer tareas comunes (contactar con la base de datos), mucho más engorrosas.

Calidad del resultado JSP tiene todo lo necesario para realizar una web de calidad.

[9] PHP Usage Stats - `http://www.php.net/usage.php`

Disponibilidad en servidores comunes La instalación y configuración de un servidor típico apache para funcionar con JSP's suele requerir más trabajo que con PHP, pues no suele instalarse de forma automática, y requiere de trabajo extra por parte del administrador. Apache Tomcat[10] es implementación oficial de JSP y Java Servlets, y suele ser bastante fácil de encontrar para las respectivas distribuciones.

Otras motivaciones El autor ya ha realizado un proyecto utilizando JSP y Java Servlet, si bien no en profundidad, ya conoce las características principales. Sería interesante (o casi imprescindible) si se hubiera decidido usar un Applet Java para el desarrollo de la Pizarra.

Viabilidad Este proyecto sería totalmente viable utilizando JSP y Servlets Java, incluso si se utiliza Javascript o Flash para la implementación de las Pizarras.

Ruby

Ruby es un lenguaje que está en auge desde hace relatiamente poco, por numerosas razones. La más importante de ellas sea probablemente la plataforma Rails[11] (de ahí el conocido Ruby on Rails), que simplifica el proceso de generación de código de forma drástica. Existen numerosos ejemplos en forma de screencasts demostrando la implementación de aplicaciones simples en tiempos menores a 15 minutos[12].

Facilidad de implementación Ésta es la característica estrella de Ruby on Rails. Simplifica todas las tareas repetitivas a la hora de desarrollar aplicaciones web con un fondo de base de datos, es decir, la mayoría. Lo único que puede dificultar la implementación es la falta de experiencia del autor con esta plataforma.

Calidad del resultado El resultado tendrá la misma calidad que el realizado por cualquier otro lenguaje de scripting. Hay argumentos que defienden la mayor calidad de las webs realizadas en ruby por su grandísima simplicidad, lo cual ayuda a una futura expansión mucho más sencilla.

Disponibilidad en servidores comunes Ruby on Rails se puede considerar hoy día como uno de los standards del desarrollo web, y por tanto cada vez más y más hostings ofrecen soporte para el mismo. Existen múltiples formas de tener una aplicación Ruby on Rails funcionando en un servidor, apareciendo cada vez opciones más sencillas y completas. Por ejemplo, [13], también llamado `mod_rails` es un módulo para Apache que promete simplificar el proceso puesta online de aplicaciones a algo tan sencillo como el de cualquier aplicación en `PHP`.

Otras motivaciones Hoy por hoy existe una gran expectación en cuanto a Ruby on Rails. Cada vez más webs se desarrollan con esta tecnología, y eso se demuestra en que cada vez más se pueden encontrar empresas de hosting con soporte para Rails. No hay ninguna razón objetiva para apoyar todas estas consideraciones, es posible que Rails no deje de ser una plataforma minoritaria, pero a día de hoy, tiene un crecimiento muy marcado que lo convierte en una plataforma muy atractiva a la hora de profundizar los conocimientos del autor en temas de desarrollo web.

Viabilidad No hay ninguna razón para que este proyecto no sea viable bajo Ruby.

[10] Apache Tomcat - `http://tomcat.apache.org`
[11] Ruby on Rails - `http://www.rubyonrails.org/`
[12] Screencasts of Ruby on Rails, Creating a weblog in 15 minutes - `http://www.rubyonrails.org/screencasts`
[13] Phusion Passenger - `http://www.modrails.com/`

Conclusión

En este apartado las tres opciones que se han considerado son prácticamente igual de competentes. Si bien Ruby tiene más posibilidades de conseguir un desarrollo más rápido y de mayor calidad, es también el que necesitará más tiempo de aprendizaje. Debido a que las tres opciones son prácticamente equivalentes en cuanto a viabilidad se refiere, Ruby será en lenguaje escogido para este proyecto, como principal razón el interés del autor en estudiar áreas nuevas del desarrollo web.

1.2.3. Entendiendo las Tecnologías

El desarrollo de una página web no se puede tomar como el desarrollo de una aplicación cualquiera. Para este proyecto se han diferenciado dos partes que deberán seguir un desarrollo independiente antes de juntarse. El desarrollo de la pizarra si puede plantearse de forma similar a una aplicación, siempre teniendo en cuenta las grandes limitaciones que javascript impone. Es por eso que se considera importante entender lo máximo posible el lenguaje antes de empezarse a plantear como será el proceso de diseño, y posterior implementación.

No hay que olvidar tampoco, que gracias a que Ruby, o más concretamente Rails, ofrece una gran cantidad de opciones para automatizar tareas comunes en el desarrollo web, incluyendo pequeños trozos de javascript, también se intentará entender cual es el funcionamiento de Ruby, qué oportunidades ofrece Rails, y cómo puede ayudar en el desarrollo de la Pizarra.

En cuanto al desarrollo del sistema de la web, se seguirán los procesos típicos para ello, en este caso enfocado a Ruby on Rails, y las facilidades que nos aporta. Ruby on Rails, al fin y al cabo, considera que revoluciona la forma en que se desarrollan webs hoy día [14], por tanto conviene ver de qué tipo de revolución se está hablando.

Javascript

Las características básicas se han descrito anteriormente, pero es necesario entender más profundamente como funciona, y cual es el proceso típico de desarrollo de un javascript, para poder planear el desarrollo de la Pizarra. Se quiere evitar empezar a escribir código ciegamente sin tener un entendimiento previo de qué se está manejando.

Mozilla Developer Center contiene un artículo[15] donde se explican todas las características de orientación a objetos en javascript, y como implementarlas con un ejemplo muy claro. Tiene una forma un tanto peculiar de crear y heredar elementos, pero es posible trabajar sin problemas, teniendo en cuenta que javascript no soporta herencias múltiples.

Document Object Model o DOM, como es conocido más habitualmente, son una serie de objetos que ofrecen los navegadores cuando se está ejecutando código javascript. Aquí es donde residen la mayoría de problemas de compatibilidad entre navegadores, puesto que el lenguaje en si es interpretado de igual manera por todos ellos, pero el DOM que ofrecen no es siempre equivalente, lo cual produce situaciones problemáticas.

Con dichos objetos se puede acceder a todos los aspectos del navegador y de la web que se está mostrando, tanto para consultarlos como para modificarlos. Así, por ejemplo, el objeto `document` permite cambiar el código web que se está mostrando, para por ejemplo, mostrar u ocultar partes, consultar el texto escrito en un formulario, etc; o el objeto `window` permite consultar las cosas referentes con el navegador, de forma que podemos cambiar el comportamiento

[14] Quotes about Ruby on Rails - `http://www.rubyonrails.org/quotes`

[15] Introduction to Object-Oriented JavaScript - `http://developer.mozilla.org/en/docs/Introduction_to_Object-Oriented_JavaScript`

de distintos eventos relacionados con el ratón o el teclado, o mostrar avisos en forma de ventanas de confirmación.

Librería gráfica Es importante recordar que Javascript es un lenguaje de scripting sin ningún tipo de soporte para gráficos incorporados. Todo lo que se puede hacer es, por ejemplo, modificar el código fuente de la web mediante javascript y diferentes eventos (de ratón o teclado). Se tiene que buscar, por tanto, una forma de mostrar gráficos modificando código que pueda estar dentro de una web.

Los elementos que se quieren dibujar para este proyecto son perfectamente implementables por lo que se conoce como gráficos vectoriales. Círculos, líneas, cuadrados, texto, todo esto se puede hacer de forma sencilla con cualquier programa que permita editar este tipo de gráficos. Existe un formato abierto llamado SVG[16], cuyo formato no es binario como suelen ser los propietarios, sino especificado en XML. Gracias a esto, es posible crear y modificar gráficos vectoriales mediante Javascript. Un ejemplo de archivo SVG podría ser el de la figura 1.3 (ejemplo extraido de Wikipedia)

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
                     "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="467" height="462">

  <!-- This is for the red square -->
  <rect x="80" y="60" width="250" height="250" rx="20" fill="red"
        stroke="black" stroke-width="2px" />
  <!-- This is for the blue square -->
  <rect x="140" y="120" width="250" height="250" rx="40" fill="blue"
       fill-opacity="0.7" stroke="black" stroke-width="2px" />
</svg>
```

Figura 1.3: Ejemplo simple de archivo svg

Por desgracia, realizando pruebas sobre cómo se incluyen gráficos SVG en páginas web, se observa que dichos gráficos no funcionan en ninguna versión de Internet Explorer (Firefox, Safari y Opera funcionan sin problemas). Se descubre que es necesario un plugin para poder visualizar dichos archivos en este navegador, pero que no obstante, existe otro formato llamado VML[17], que si está implementado en las versiones actuales de Internet Explorer. El siguiente párrafo introductorio de la entrada sobre VML en la Wikipedia[18] explica perfectamente porqué no funcionan los SVG en Internet Explorer:

[16] Scalable Vector Graphics - `http://www.w3.org/Graphics/SVG/`
[17] Vector Markup Language - `http://www.w3.org/TR/1998/NOTE-VML-19980513`
[18] Vector Markup Language, Wikipedia - `http://en.wikipedia.org/wiki/Vector_Markup_Language`

> **Vector Markup Language (VML)** is an XML language used to produce vector graphics. VML was submitted as a proposed standard to the W3C in 1998 by Microsoft, Macromedia, and others, but was rejected as a web standard because Adobe, Sun, and others submitted a competing proposal known as PGML. The two standards were joined to create SVG.
>
> Even though rejected as a standard by the W3C, and largely ignored by developers, Microsoft still implemented VML into Internet Explorer 5.0 and higher and in Microsoft Office 2000 and higher.

A pesar de no ser la solución ideal, todos los Internet Explorer con versión 5.5 o superior implementan este lenguaje, con un formato similar al SVG. Es posible, por tanto, tratar con gráficos vectoriales en los navegadores mayoritarios (los que interesa en este proyecto), gracias a estos dos formatos.

Canvas Estas dos no son las únicas formas de tratar con gráficos vectoriales en páginas web. En la nueva especificación HTML (versión 5[19], actualmente aún en formato borrador aún), se especifica un nuevo elemento, denominado `<canvas>`, y cuyo objetivo es la representación gráficos vectoriales directamente en la web. Es independiente de VML o SVG, y ya está implementado en los navegadores mayoritarios, excepto Internet Explorer. Existe, sin embargo, una librería en javascript[20], que automáticamente transforma cualquier elemento `<canvas>` en su equivalente en VML, por lo cual se puede considerar que es soportado en todos los exploradores mayoritarios actuales.

El funcionamiento de canvas es completamente distinto al de SVG y VML. Se basa en considerar que hay un puntero que puede ir haciendo diversos tipos de trazos. Se puede mover a cualquier punto dentro de un área definida, y moverse hacia otro haciendo el dibujo. Existen funciones para hacer todo tipo de "trazos", permitiendo dibujar líneas, círculos, y todo lo necesario.

Este tipo de planteamiento, sin embargo, no es adecuado para el tipo de aplicación que se pretende desarrollar. En el caso de la pizarra, existirán diferentes elementos que se irán añadiendo, quitando y modificando. El formato SVG o VML es ideal, puesto que está formado por dichos elementos, y para por ejemplo, crear un círculo, simplemente se añade una nueva etiqueta correspondiente a un círculo. En el caso de Canvas, sin embargo, habría que redibujar todo desde el principio, incluyendo el trazo final de un círculo. Básicamente, cada vez que se modifique algo en canvas, hay que redibujar todo de nuevo.

Por lo tanto, canvas es ideal para realiar dibujos estáticos, pero poco adecuado para realizar aplicaciones interactivas como será ésta.

Ruby

Ruby es un lenguaje inspirado en Perl, Smalltalk, Eiffel, Ada y Lisp, creado por Yukihiro "matz" Matsumoto[21]. Ruby en si es solo el lenguaje, y existen diferentes implementaciones. De ellas, la oficial está implementada en `C` y es **interpretado en una sola pasada**[22]. Típicamente se puede ejecutar un archivo ruby desde la línea de comandos, con una linea parecida a la siguiente:

```
$ ruby archivo.rb
```

Desarrollando la web, Ruby on Rails Rails[23] es una plataforma de desarrollo (framework) web basada en Ruby, que ayuda a desarrollar aplicaciones web, y que sigue el paradigma Modelo

[19] W3C, HTML5 - `http://www.w3.org/html/wg/html5/`
[20] Explorer Canvas - `http://excanvas.sourceforge.net/`
[21] Acerca de Ruby - `http://www.ruby-lang.org/es/about/`
[22] Ruby, Wikipedia - `http://es.wikipedia.org/wiki/Ruby`
[23] Ruby on Rails - `http://www.rubyonrails.org/`

Vista Controlador. Dicho patrón es muy semejante al conocido patrón en 3 capas, separando la capa que trata con los datos, la que realiza operaciones, y la del interfaz. Gracias a este enfoque es posible por fin enfocar el desarrollo de una aplicación web de forma muy similar al desarrollo de cualquier otra aplicación tradicional como la que se está acostumbrado. Se tendrán las clases del dominio, que representarán objetos bien definidos, que serán usados por los distintos controladores, cada uno de los cuales agrupará una serie de funcionalidades comunes y afines, y todo esto manejado desde las diferentes vistas.

Hasta ahora el proceso de desarrollo de una web era una tarea bastante "artesanal", por el hecho de que estaba todo agrupado en una sola capa. Programar de forma tradicional en `PHP` es como trabajar con solo la capa de Vistas de Ruby on Rails.

Instalando Ruby Uno de los requisitos establecidos es que fuera relativamente sencillo de instalar en un servidor típico web con Apache y MySQL. Para poder realizar pruebas a lo largo de todo el proceso de desarrollo en un entorno lo más realista posible, se ha contratado un servicio de hosting de tipo VPS, en slicehost[24]. Este tipo de servicio de hosting ofrece un entorno virtual privado (Virtual Private Server) que a efectos prácticos significa tener un entorno Linux con la distribución de tu elección, con unos recursos asgurados. La diferencia con los sistemas de hosting compartidos es que, aunque estés compartiendo la máquina con otras personas, al estar todos en entornos virtualizados con unos recursos reservados, no hay posibilidad de unos usuarios acaparando los recursos.

Esto también significa un entorno de pruebas excelente para los propósitos de este proyecto. Este servicio de hosting ofrece una serie de artículos para ayudar a la instalación de todo tipo de configuraciones para Ruby on Rails. Tomando como partida la distribución **Ubuntu Hardy**, y siguiendo los pasos iniciales básicos de configuración [25] [26], solo queda instalar Ruby[27], Apache[28], MySQL[29], y finalmente Phusion Passenger[30] (aunque podría ser cualquier de las otras opciones, como Mongrels o Thin). A pesar de parecer una instalación bastante larga, si se tomara como base un servidor con Apache y MySQL funcionando, solo se tendría que añadir el soporte para `ruby`, `rubygems`, `rails` y phusion passenger (`mod_rails`).

1.2.4. Otras consideraciones

A lo largo de este documento no se ha tratado el tema de base de datos. Ante todas las opciones disponibles, se pueden descartar todas las que no sean libres, por un coste de licencias generalmente inalcanzable en el ámbito de este proyecto (no hay ningún tipo de subvención). De entre las opciones libres la base de datos por excelencia es MySQL[31], que es considerada el compañero ideal para Ruby. A simple vista no existen requisitos extras que puedan hacer considerar alguna otra opción, por tanto se da por hecho que MySQL será el Sistema Gestor de Base de Datos de este proyecto.

[24] `http://www.slicehost.com/`
[25] `http://articles.slicehost.com/2008/4/25/ubuntu-hardy-setup-page-1`
[26] `http://articles.slicehost.com/2008/4/25/ubuntu-hardy-setup-page-2`
[27] `http://articles.slicehost.com/2008/4/30/ubuntu-hardy-ruby-on-rails`
[28] `http://articles.slicehost.com/2008/4/25/ubuntu-hardy-installing-apache-and-php5`
[29] `http://articles.slicehost.com/2008/7/8/ubuntu-hardy-installing-mysql-with-rails-and-php-options`
[30] `http://articles.slicehost.com/2008/5/1/ubuntu-hardy-mod_rails-installation`
[31] MySQL - `http://www.mysql.com/`

1.3. Diseño

Como buen proyecto de desarrollo de una aplicación, se pretende seguir los pasos naturales comunes a todos los proyectos de este tipo. Primero se establecieron una serie de criterios en la etapa de especificación. Seguidamente hubo un proceso de comprensión de las tecnologías para ver las diferencias en el proceso de desarrollo de una aplicación web respecto a las aplicaciones más clásicas. Y finalmente solo queda realizar una etapa de diseño que, en mayor o menor medida, ayude a planear un desarrollo estructurado de esta aplicación.

Como en etapas anteriores, también es posible enfocar el diseño de esta aplicación como dos elementos bastante diferenciados, y que necesitarán de un proceso de desarrollo suficientemente independiente como para ser considerados por separado. Es cierto que deberán interactuar, y por eso se incluye un apartado que pretende entender de qué forma se llevará a cabo esta comunicación, y de qué manera afectará al enfoque que se le deba dar en cuanto al desarrollo.

1.3.1. Pizarra

Se ha explicado con anterioridad las características de Javascript, el cual, a pesar de sus limitaciones, es orientado a objetos, y es posible programar de forma bastante modular gracias a ello. Un boceto inicial de lo que podría ser el diagrama de clases del dominio se puede encontrar en la figura 1.4.

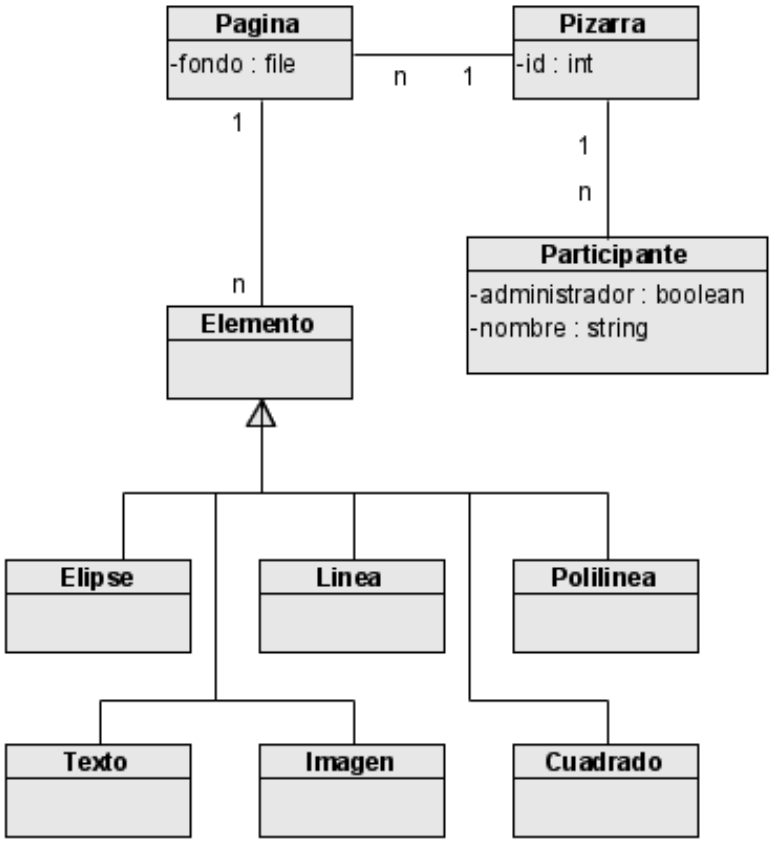

Figura 1.4: Boceto de clases del dominio

Cada Pizarra es única, tiene su identificador, y en la que participan una serie de usuarios. Dicha Pizarra tiene una serie de páginas, y un fondo opcional. Este fondo se generará si el creador

de la pizarra ha subido un documento que sirva de fondo (recordando, un PDF, powerpoint, o simplemente una serie de imágenes). A partir de ahí, simplemente habrá una serie de elementos, que podrán ser de distinto tipo. Es difícil de definir en estos momentos qué atributos deben tener estos elementos, pues dependen en gran medida de las forma en que están implementados en SVG o VML.

Hay una parte importante de la especificación de las funciones de la pizarra que se ha considerado opcional, y por eso no aparece aquí. Por ejemplo, si se llegara a implementar un chat, y se guardara registro, harían falta clases extra. En cualquier caso, este diagrama da una buena idea de los elementos que se manejarán a lo largo del uso de la Pizarra.

Una vez sabido esto, el código que manejará estos elementos se distribuirá en una serie de módulos. Hay que recordar que la Pizarra está integrada en un sistema, que es la página web, que le proveerá con la información necesaria. Por ejemplo, Javascript de por si no es capaz de consultar una base de datos, pero si consultar otra página mediante Ajax, que le devuelva lo que necesita saber. Dicha página será generada por el Sistema, que en este caso funcionará bajo Ruby on Rails, que será el que consultará la base de datos y genere los datos con el formato adecuado para que Javascript lo pueda entender.

Renderizado Aquí se agruparán las funciones que dibujen los elementos. Debe tener las funciones necesarias para poder dibujar los elementos que se soporten (líneas, polilíneas, cuadrados, círculos, etc), así como editarlos, y que sea capaz de hacerlo independientemente del navegador que se esté usando.

Dibujo Será el módulo que controle los eventos del ponente. Debe ser capaz de entender los movimientos de ratón y teclado de forma que indique al módulo de renderizado las figuras que debe crear de forma que se experimente una sensación de dibujo interactivo. Además, debe de comunicar al sistema cuándo se ha creado, editado, o eliminado algún elemento.

Comunicación Éste módulo incluirá las funcionalidades de comunicación entre la interfaz, en javascript (recordando que javascript es incapaz de consultar bases de datos), y el sistema web. Será capaz de informar en ambas direcciones, tanto cuando el ponente ha modificado la pizarra para decírselo al sistema, como cuándo el sistema recive nuevos elementos, para comunicárselos a los espectadores.

Controlador de Usuarios Debe de controlar los usuarios que el sistema le dice que entran a la Pizarra, y darles los privilegios adecuados. Debe de mostrarlos en la pizarra para informar a los demás.

Debido a las características peculiares de Javascript, se considera este enfoque como el más sencillo y eficiente para un programador con conocimientos escasos del lenguaje. El hecho de que javascript sea orientado a objetos, permite que se creen clases adecuadas para por ejemplo representar las clases del dominio, sin embargo, sigue siendo más sencillo implementar dichos controladores como una serie de funciones agrupadas en un archivo, más que hacer realmente una clase Renderizado, con subfunciones. En cualquier caso, se dará una gran importancia a intentar implementar de forma coherente con el paradigma de orientación a objetos, en la medida de lo posible acorde con las características de Javascript, y el nivel de conocimiento del lenguaje del programador. En esta fase del desarrollo es difícil definir hasta qué punto será posible.

1.3.2. Sistema

El sistema debe servir como intermediario entre todos los usuarios que participan en una pizarra, y ser capaz de realizar las funciones que Javascript, de por si solo, no es capaz. Antes de

empezar a participar en alguna pizarra hay una serie de acciones que deben de ser controladas por el sistema. Dichas funcionalidades se pueden recoger en una serie de módulos, los cuales más adelante se intentarán formalizar en una estructura compatible con el paradigma Modelo-Vita-Controlador de Rails.

Control de Usuarios y Grupos Es necesario mantener un control de los usuarios y los grupos a los que se pertenece, y para ello es necesario una base de datos. Los usuarios tienen que poder registrarse y hacer las funciones típicas, como Login, edición de datos, creación de grupos, etc. Éste módulo debería ser capaz de controlar todo lo referente a usuarios y grupos según el comportamiento que se ha definido anteriormente.

Control de Pizarras Las responsabilidades de este módulo serían las de mantener un control de las pizarras que existen, sus permisos, y su contenido. Cada pizarra tiene una serie de páginas, y cada página una serie de elementos, que deben de ser accesibles y modificables, así como poder crear o eliminar nuevos.

Comunicación con Javascript Ésta es la funcionalidad más importante, y a falta de un nombre más apropiado, este módulo debe hacer precisamente esto, ser capaz de comunicarse con la interfaz, que se está ejecutando en el cliente, y no en el servidor. Debido a las características de Ajax y de Rails, se cree que no será posible separar formalmente estas funcionalidades, y que deberá ser incluida en los otros dos módulos. La sección siguiente ayudará a entender porqué.

Entendiendo AJAX

Antes de seguir se considera interesante entender como funciona Ajax, pues es la única técnica conocida que permite la interactividad que se espera de un programa como el que se está desarrollando, pero a su vez hace que se deban enfocar las cosas de una forma un poco distinta al paradigma típico de Cliente-Servidor.

Para empezar, hay que entender que la forma en que uno navega por la web, es consultando distintas páginas, una a una. La única forma de que un servidor es capaz de comunicarse con un usuario, es cuando este usuario le ha hecho alguna consulta. No existe forma alguna de que un servidor abra una conexión hacia un usuario por voluntad propia. Típicamente, cuando se navega, se consulta una página, que posiblemente dentro contenga un enlace a otra, y al clicar en dicho enlace, lo que se hace es pedir al servidor esta segunda página.

Ajax, lo que hace, es que el navegador pueda realizar este proceso de "pedir" una página, y obtener el contenido en formato texto de forma que Javascript puede hacer con él lo que quiera. Ésto elimina la necesidad de cargar una página completa, y provee una serie de beneficios que el programador puede explotar.

Ejemplo

Se quiere mostrar en una web de un programa, la cantidad de descargas que se han producido. En un escenario típico en que se use, por ejemplo, PHP, se tendría la página principal, y en el lugar donde se muestra el número de descargas se incluiría la función `numDescargas()`, mostrando el número de descargas a la hora de generar dicha página.

Sin embargo, se quiere modernizar dicha página, y hacer que dicho número se modifique en tiempo real. Para ello se incluye un pequeño código en javascript en el lugar donde antes estaba la funcion `numDescargas()`, que lo que hará será hacer constantes consultas, pero esta vez mediante Ajax.

Simplificando en gran medida, se supone que se tiene implementada la función `ajax("pagina.php");`, la cual retorna el contenido del `archivo.php` que se le pasa como parámetro, en formato texto.

Dentro del archivo `pagina.php` tendríamos un código como el siguiente:

```
<?php numDescargas() ?>
```

Ajax, al consultar este archivo, lo hace desde el cliente, es Javascript, es el navegador el que lo está ejecutando, por tanto, el servidor lo ve como una consulta externa, y por tanto, ejecuta el código php. Si se accediera al archivo `pagina.php` desde el navegador, veríamos simplemente un número en pantalla. Por tanto, lo que la funcion `ajax()` devuelve no es el texto "`<?php numDescargas() ?>`", sino el número de descargas, es decir, lo que devuelve esa función.

Una vez se tiene dicho número en una variable en javascript, es trivial modificar el campo donde debería aparecer. Considerando un trozo de código como el siguiente:

```
<div><span id="descargas"></span> descargas y subiendo!</div>
```

Se podría modificar mediante una línea como la siguiente, dentro de un bucle que no acabe nunca:

```
while(true) document.getElementById("descargas").innerHTML = ajax("pagina.php");
```

Hay que tener muchos más factores en cuenta, como que hay que esperar a que el servidor conteste a la consulta, pero a modo de ejemplo se considera suficiente.

Se entiende, por tanto, que para que Ajax sea posible hacen falta una serie de páginas que puedan "pasar" datos a Javascript en un formato que Javascript pueda entender.

Estructura de Rails

A la hora de manejar usuarios y grupos, es un proceso natural de Rails, pues es una aplicación típica de uso de bases de datos en páginas web. Los modelos se definirán con los atributos apropiados a cada caso, con sus respectivos controladores, y vistas para poder manejarlo todo. Existen, de hecho, plugins que permiten implementar todas las funcionalidades de registro de usuarios de forma sencilla. Sin embargo, se deben añadir una serie de vistas *extra* que al consultarlas no devuelvan la página entera, sinó simplemente el texto adecuado para que Javascript lo entienda. Es en estas vistas donde se implementarán las funcionalidades que idealmente deberían haber ido en el módulo denominado anteriormente como "Comunicación".

En cuanto a las Pizarras, es muy semejante. Una vez definido un modelo de datos de todos los contenidos de una pizarra, mediante el controlador y vistas adecuadas, se puede consultar cualquier elemento de forma sencilla. En el caso de las pizarras, todas las vistas excepto la principal, serán *Javascript compatibles*, pues en el único caso en que se modifiquen los contenidos de una pizarra será mediante la interfaz que se habrá preparado.

1.3.3. Consideraciones finales

Se recuerda que este capítulo trata de el trabajo previo realizado antes de coeenzar a trabajar con dos lenguajes prácticamente desconocidos previamente por el autor. Debido a eso, y a las diferencias esenciales entre una página web y una pieza de software tradicional, se considera imposible profundizar más en la formalización del diseño.

En cualquier caso, se considera que se ha definido de forma completa e inconfusa las funcionalidades que deberá cumplir el sistema y se han asignado responsabilidades a las dos *capas* que formarán la página web (interfaz y sistema), habiendo obtenido para ello, un conocimiento previo que ha permitido realizar dichas asignaciones de forma coherente con la realidad de un proyecto web como éste.

En capítulos siguientes será cuando se explique de forma más profunda cuales han sido las soluciones que se han aplicado en cada etapa, habiendo seguido las directivas establecidas en este primer paso en la medida de lo posible, con reflexión sobre la adecuidad de las mismas.