Adaptación de una aplicación RIA desarrollada en Flex a una aplicación HTML5

Proyecto Fin de Carrera

10 de julio de 2012

Imanol Luengo Muntión

Director:
Julián Gutiérrez Serrano

Ingeniería Superior en Informática

Facultad de Informática de San Sebastián (FISS)

Universidad del País Vasco
Euskal Herriko Unibertsitatea

© 2012 Imanol Luengo

Agradecimientos

Quiero dar las gracias a la gente que me ha estado apoyando a lo largo del proyecto. A Juanan Pereira e Inko Perurena, que han estado debatiendo conmigo y ayudándome a sacar el proyecto adelante con dudas, consejos y críticas: todo ello ha hecho que el proyecto termine en mejor puerto. A mis amigos, que aunque no lo hayan hecho conscientemente, han respetado mis ausencias a lo largo del año en épocas puntuales: por estudios, trabajo y proyecto. A mi familia, que un año más han confiado en mi y me han apoyado en la toma de decisiones sin ponerme muchas pegas. Y, por último, a Naroa, una persona muy especial que ha estado a mi lado apoyándome a lo largo del año con todo lo que necesitara.
A todos, gracias.

II

Índice general

Índice de figuras

Índice de cuadros

Resumen

Babelium Project (también BP de aquí en adelante), es un proyecto de código abierto cuyo principal objetivo es fomentar el aprendizaje de idiomas online. Para ello, haciendo uso de distintas tecnologías, se creó una aplicación web, *Rich Internet Applications (RIA)*, que permite a los usuarios practicar idiomas por práctica oral. La aplicación dispone de varios módulos donde, gracias al streaming de vídeo, los usuarios pueden grabar o evaluar ejercicios de forma colaborativa.

Babelium Project empezó siendo una aplicación desarrollada bajo el conjunto de tecnologías Flex, un conjunto de tecnologías de Adobe cuya compilación resultante es un aplicación (web en este caso) basada en Flash. Sin embargo, con la reciente llegada de HTML5, todo parece indicar que Adobe abandonará Flex en un futuro [1] [2] para centrar sus esfuerzos en el desarrollo de soluciones para esta nueva tecnología. Por esa razón, nació este proyecto, con el fin de migrar Babelium Project a HTML5, conjunto de tecnologías con gran futuro y acogida en el mundo web.

Mi trabajo en este proyecto ha consistido, principalmente, en analizar la factibilidad y proceso adecuado de la migración de un proyecto de dimensiones considerables desarrollado bajo Flex al conjunto de tecnologías que forman HTML5, teniendo para ello el proyecto Babelium como caso de prueba. Las fases principales del proyecto han sido: análisis del estado de HTML5, análisis de factibilidad, elección de un conjunto de tecnologías para la migración, desarrollo de patrones de migración y, por último, migración de Babelium utilizando dichas tecnologías y siguiendo dichos patrones.

Palabras clave: migración, Flex, HTML5, factibilidad, patrones, prototipo.

[1] Adobe abandonará Flash en dispositivos móviles para centrarse en HTML5: [1]
[2] Adobe dona Flex a la comunidad: [2]

Capítulo 1

Definiciones y acrónimos

Agile: metodologías ágiles de desarrollo de software. Se centran en las funcionalidades y potencian la productividad.

AJAX: *Asynchronous Javascript And XML*. Conjunto de tecnologías para la carga dinámica de contenidos web.

API: *Application Programming Interface*. Lista de métodos y propiedades de una implementación para facilitar (incluso a terceros) su uso o reutilización.

Cairngorm: framework que ofrece abstracción al cliente de la aplicación implementando el patrón MVC.

CMS: *Content Management System*. Sistema de gestión de contenidos para la web.

CSS3: es un lenguaje usado para definir la presentación (estructura y estilo) de un documento estructurado escrito en HTML ó XML (y por extensión en XHTML).

Delegate: Otro patrón de programación utilizado en el proyecto. Consiste en delegar acciones que requieran de información de terceros (base de datos, conexión con servicios remotos) a clases externas, especializadas en ello.

Flex: Tecnología basada en MXML y ActionScript para la creación de un cliente web. Genera un archivo SWF (Shockwave Flash).

Gateway: puerta de enlace entre 2 sistemas.

GUI: *Graphical User Interface*, interfaz gráfica de usuario.

HTML5: (*HyperText Markup Language*, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML.

Iteración: periodos en los que se divide el ciclo de vida de un proyecto. Cada iteración tiene una función distinta.

MVC: patrón de programación Modelo-Vista-Controlador, que pretende abstraer a las clases Vista (Visualización del programa) de la compleja carga o calculo de información a mostrar.

Parseo: se dice parseo del análisis gramatical de un texto con el fin de llevar a cabo acciones en función del resultado obtenido.

RIA: *Rich Internet Applications*. Se refiere al enriquecimiento de aplicaciones web mediante diversas tecnologías, haciendo posible ejecutar vía web aplicaciones antes sólo disponibles para escritorio.

Stories: conjunto de subapartados en los que se divide cada una de las iteraciones del proyecto, dividiendo de esta manera una iteración compleja y abstracta en agrupaciones más concretas y fáciles de manejar.

Widget: En informática, un widget es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine (fuente en Wikipedia).

Capítulo 2

Introducción

Contents

Este trabajo pretende analizar la factibilidad y dar solución a la migración de proyectos de tecnologías Flex al conjunto de tecnologías que forman HTML5, teniendo Babelium Project como caso de prueba. Así mismo, el proyecto pretende ser un ejemplo a seguir, o tener en cuenta, para otros desarrolladores que pretendan migrar una aplicación de características similares.

> *Es posible que el lector de esta memoria no esté familiarizado con el proyecto o el funcionamiento de Babelium Project. Si éste fuera el caso, se recomienda encarecidamente leer el apéndice D antes de proseguir.*

Con el fin de favorecer la lectura y comprensión del trabajo realizado, se ha escogido para este documento una estructura dirigida a un cliente de un proyecto informático, que no tiene por qué ser un experto en el tema. Por ello, se hará más hincapié en la narración de los objetivos conseguidos y decisiones tomadas, que en la exposición de pesados diagramas y demostraciones. De este modo, cualquier persona, aunque no tenga grandes conocimientos, puede hacerse una idea rápida del trabajo realizado y su importancia simplemente leyendo el índice e introducción de cada tema, dejando para los interesados en los detalles técnicos los subapartados más específicos y sobre todo, los apéndices con datos técnicos e información y guías para los desarrolladores.

Dicho esto, los apartados principales mostrarán una explicación lo más abstracta y amena posible del tema en cuestión, desde el punto de vista de un usuario (o desarrollador) externo, mientras que el apéndice contendrá toda la información técnica de la implementación de las soluciones, para los que deseen profundizar en el tema.

Esta estructura facilita la evaluación del trabajo, frente a otras estructuras más generales que no reflejan independencia entre las distintas tareas realizadas, y mucho menos entre los detalles dirigidos a lectores comunes y desarrolladores experimentados e interesados en profundizar en el tema.

A lo largo de las siguientes páginas, se mostrarán los objetivos del proyecto, el control de la planificación y la gestión del mismo, así como los detalles técnicos del análisis y diseño de la solución propuesta. Además, se hará referencia a las distintas tecnologías y metodologías usadas, y a las buenas prácticas llevadas a cabo para ofrecer una solución elegante a un problema complejo.

2.1. Situación Inicial

Babelium Project llevaba aproximadamente 3-4 años de desarrollo, años en los cuales distintos estudiantes e investigadores de la facultad de informática estuvieron trabajando

para que Babelium Project pudiera ver la luz[1]. Como proyecto de la *Ingeniería Técnica en Informática de Sistemas (ITIS)* yo mismo estuve trabajando dentro del grupo GHyM y desarrollé ahí mi PFC (*Proyecto de Fin de Carrera*) [3] unificando y modularizando el código de Babelium para ofrecerle mayor escalabilidad, además de añadir nuevas funcionalidades y revisiones de seguridad.

Ese mismo año, en 2009-2010, trabajamos dentro del proyecto como estudiantes de informática Beñat Lizarazu, Iñigo Escamochero, Jonathan Larrinaga, Iker Jericó y yo, y como investigadores Inko Perurena, Juanan Pereira y Silvia Sanz, dando como resultado la primera versión estable y pública de Babelium Project, desarrollada bajo Flex.

Tras terminar mi PFC, seguí en contacto tanto con el grupo de investigación como con mi anterior tutor, Juanan Pereira, y seguí echando una mano en el proyecto a modo de soporte técnico y solucionando dudas/bugs[2] puntuales, a la vez que inicié la Ingeniería Superior en Informática.

Por cuestiones personales preví que mis cursos académicos 2010-2012 iban a estar un poco saturados de estudios y trabajo, por lo que, a comienzos del 2011, contacté con Juanan Pereira para apalabrar un PFC, para la Ingeniería Superior en Informática, a modo de ampliación dentro de Babelium Project; para poder ir investigando poco a poco y poder entregar un proyecto de calidad.

Tras debatir sobre el camino que Babelium Project debería seguir, decidimos que la prioridad era migrar Babelium Project al conjunto de tecnologías HTML5, que estaba empezando a ver la luz por aquel entonces, y, con Julian Gutiérrez como tutor del proyecto, definimos las principales fases que, a *grosso modo*, iba a tener el proyecto:

- ➔ Investigación del estado de HTML5 y soporte de sus funcionalidades
- ➔ Comprobar la factibilidad de la migración de un proyecto Flex a HTML5
- ➔ Migrar Babelium Project de Flex a HTML5 como caso de prueba

2.2. Razones de la migración

Con la reciente salida del estándar HTML5 y el presunto futuro de Flex [3] [4], decidimos que el futuro de Babelium no era seguir desarrollando funcionalidades sobre Flex, si no

[1] http://www.babeliumproject.com
[2] Errores de programación de la aplicación
[3] Adobe abandonará Flash en dispositivos móviles para centrarse en HTML5: [1]
[4] Adobe dona Flex a la comunidad: [2]

empezar a migrarlas a HTML5, la tecnología web del futuro. Cómo se puede ver en la figura 2.1 [4], HTML5 pretende ser un estándar global para el desarrollo de *Rich Internet Applications* que hasta entonces había que desarrollar utilizando tecnologías privativas y dependientes de terceros, que requieren distintos plugins para poder funcionar; como *Active X* de Microsoft o *Flash Player* de Adobe.

Figura 2.1: Áreas que abarca el conjunto de tecnologías que conforman HTML5

Migrar a HTML5 supondría una gran ventaja para Babelium Project, porque su software dejaría de depender del trayecto de una compañia, como era Adobe con Flex. HTML5 como estándar en tecnología web, permitiría, en un futuro, que Babelium Project pudiera ser soportado en un mayor número de dispositivos: tablets, móviles, ordenadores... de distintas arquitecturas y sistemas operativos; sin mencionar que las tecnologías libres y estándares están más documentadas y soportadas por la comunidad de usuarios[5]. Por ejemplo, nada más salir HTML5 ya se crearon distintos portales para fomentar su uso y explotación, como por ejemplo *HTML5 Rocks* [5] y *Mozilla HTML5 Demos* [6], entre otros muchos.

[5]Conjunto de usuarios y desarrolladores interesados en las tecnologías de HTML5 que ayudan de forma altruista con documentación, dudas, ejemplos y/o guías y tutoriales.

2.3. Necesidades de la migración

Pese a lo mencionado en el apartado anterior, y por desgracia, HTML5 es un estándar demasiado reciente y tanto al comienzo como al final de este proyecto, sus funcionalidades aún son pobremente soportadas por todos los navegadores. Mientras que las últimas versiones de los navegadores web[6] soportan gran cantidad de funcionalidades, otros tienen aún mucho camino por delante, y eso sin contar que hoy en día aún hay usuarios que utilizan navegadores obsoletos, como Internet Explorer 6-7. Es por ello que encontrar una solución de migración a HTML5 viable y disponible para todos los usuarios, no era algo para nada trivial.

Además, Babelium Project no es una aplicación web simple, se basa en el streaming y captura de vídeo, tecnologías que en un futuro HTML5 pretende soportar, pero que a día de la redacción de esta memoria, la captura de vídeo y audio ni siquiera se ha empezado a desarrollar y no estará disponible hasta más adelante.

Por el contrario, y con vistas al futuro, una versión de Babelium Project en HTML5 se convierte en la única solución viable, dado que Flash está en decadencia, y dispositivos móviles como el iPad de Apple, por ejemplo, han dejado ya de soportarlo. HTML5 es una apuesta de futuro para intentar abarcar una cuota de mercado máxima

Para poder llevar a cabo la migración de software correctamente, desde Flex hasta HTML5, pese a las adversidades que podrían aparecer teniendo el cuenta el estado de HTML5, se definieron unas necesidades básicas que debían ser cumplidas:

2.3.1. Funcionalidades

Todas las funcionalidades de Babelium Project debían de ser soportadas en HTML5, a excepción del módulo *Subtitles* y *Upload*[7], cumpliendo:

- ➔ La migración de funcionalidades tenía que ser lo más transparente posible, de forma que cualquier desarrollador que hubiese trabajado sobre Flex, fuera capaz de añadir sin muchos problemas nuevas funcionalidades sobre HTML5.
- ➔ Los módulos principales deben poder funcionar sobre el conjunto de tecnologías escogido dentro de HTML5.
- ➔ Se debía crear un control de sesiones y registro para los usuarios.

[6] Firefox, Chrome, Internet Explorer, Safari, Opera.. como los más usados

[7] La migración de estos módulos queda fuera del alcance del proyecto debido a restricciones temporales y a que el objetivo principal del proyecto, es demostrar que la migración es viable.

2.3.2. Soporte

Tal y como se ha mencionado en esta sección, HTML5 era y es aún inestable y una versión *Beta* pobremente soportada, por ello, el proyecto tenía serias necesidades de soporte para todos los usuarios:

- ➔ Escoger conjunto de tecnologías mayormente soportadas en los navegadores.
- ➔ Garantizar la navegación y soporte de tecnologías para los navegadores más adelantados en el soporte a HTML5 (al inicio del proyecto eran: Chrome y Firefox).
- ➔ Establecer medidas de *fallback*[8] en caso de que los usuarios no dispusieran de un navegador nuevo y/o actualizado.

2.3.3. Apariencia

Para que la web llamase la atención y los usuarios quisieran volver a ella, tenía que cumplir ciertos requisitos de accesibilidad:

- ➔ La navegación entre módulos y secciones de la web tenía que ser fluida y elegante.
- ➔ La web debía mantener el mismo diseño (o similar) que el que tenía Babelium Project en su versión Flex.
- ➔ La web debía ser *user-friendly* (fácil de usar para el usuario común).

[8]Medidas para que los usuarios pudieran navegar por la web y usar sus funcionalidades aunque sus navegadores no soportasen HTML5

Capítulo 3

Objetivos

Contents

Los objetivos que cubre este proyecto se pueden dividir en distintos grupos:

- Objetivos generales del proyecto
- Objetivos genéricos de una migración de Flex a HTML5
- Objetivos concretos de la migración de Babelium Project

Los objetivos generales del proyecto, son aquellos que se acordaron con el tutor y que el proyecto debía cumplir a grandes rasgos. Los genéricos de una migración y los objetivos concretos de la migración de Babelium Project, son un desglose de los objetivos que me propuse para que dichas fases pudieran ser completadas con éxito.

3.1. Objetivos generales del proyecto

Se me encomendó, como tarea principal del proyecto, migrar Babelium Project de Flex a HTML5, tarea que puede desglosarse en los siguientes objetivos:

Objetivo 1: Analizar las posibilidades de migración de una aplicación Flex a HTML5

Objetivo 2: Migración de las funcionalidades básicas de Babelium Project de Flex a HTML5

Objetivo 3: Que el trabajo de migración sirviera de guía y/o ejemplo para otros desarrolladores con aplicaciones similares

3.2. Objetivos genéricos de una migración de Flex a HTML5

Para que el trabajo pudiera ser usado a modo de ejemplo por otros desarrolladores, me planteé los siguientes objetivos:

Objetivo 4: Desglosar la aplicación Babelium Project en funciones y *clusters* de clases[1], en función del patrón de diseño y utilidad general.

[1]Conjuntos de clases con funcionalidades similares.

Objetivo 5: Investigar equivalencias para dichos clusters de clases en HTML5.

Objetivo 6: Establecer patrones de migración de Flex a HTML5 para los clusters mencionados.

3.3. Objetivos concretos de la migración de Babelium Project

Los objetivos que impuse para la migración de Babelium Project a HTML5, con el fin de facilitar la migración y la futura adición de funcionalidades, fueron los siguientes:

Objetivo 7: La migración debía realizarse reutilizando el mayor código posible de la versión en Flex

Objetivo 8: La migración debía ser lo más transparente posible, de modo que el código y estructura de ambas tecnologías fuera lo más similar posible.

Objetivo 9: La aplicación resultante debía ser fácilmente escalable y modularizada.

Objetivo 10: La aplicación web debía tener un diseño elegante, similar al de la versión en Flex y una navegación *user-friendly* para el usuario.

Objetivo 11: Migrar prácticamente la aplicación completa.

3.4. Resultados del proyecto

La implementación de este PFC formó la base de la aplicación Mintzabel.com (proyecto subvencionado por el Gobierno Vasco) y, adicionalmente, dio como resultado la aceptación de un artículo científico para el congreso "Jornadas de Ingeniería Software y Bases de Datos 2012 (JISBD 2012)". Con lo que, tendríamos en total los siguientes resultados tras el proyecto:

- ➔ Memoria del proyecto, conteniendo toda la documentación sobre las tareas realizadas en el mismo.
- ➔ Versión de Babelium Project en HTML5[2]

[2]http://html5.babeliumproject.com/demos/html5/

- ➔ Versión de Babelium Project en HTML5 modificada para un proyecto del Gobierno Vascohttp://www.mintzabel.com
- ➔ Artículo científico "An experience migrating a Cairngorm based Rich Internet Application from Flex to HTML5" para el JISBD 2012 [7]

A lo largo de esta memoria, queda descrito cómo se han cumplido todos los objetivos, y con que resultados.

- ➔ Se ha realizado un análisis exhaustivo para encontrar facilidades de migración para un proyecto Flex a HTML5.
- ➔ Se darán además patrones de migración para que un proyecto de las mismas características (Flex+Cairngorm[3]) sea migrado de forma rápida y sencilla; cumpliendo así los objetivos de documentación y reutilización del trabajo.
- ➔ Se darán también detalles de la migración concreta de Babelium Project.

Para que el lector pueda hacerse una idea rápida del tipo de módulos que se han migrado de Flex a HTML5, y del tipo de migración que este proyecto plantea, se muestra a continuación un ejemplo gráfico del módulo de grabación de ejercicios en Flex (figura 3.1) y el resultado de su migración a HTML5 (figura 3.2).

[3]Framework que aplica el patrón MVC en Flex, más información en la sección 5.1.1 y 7.4

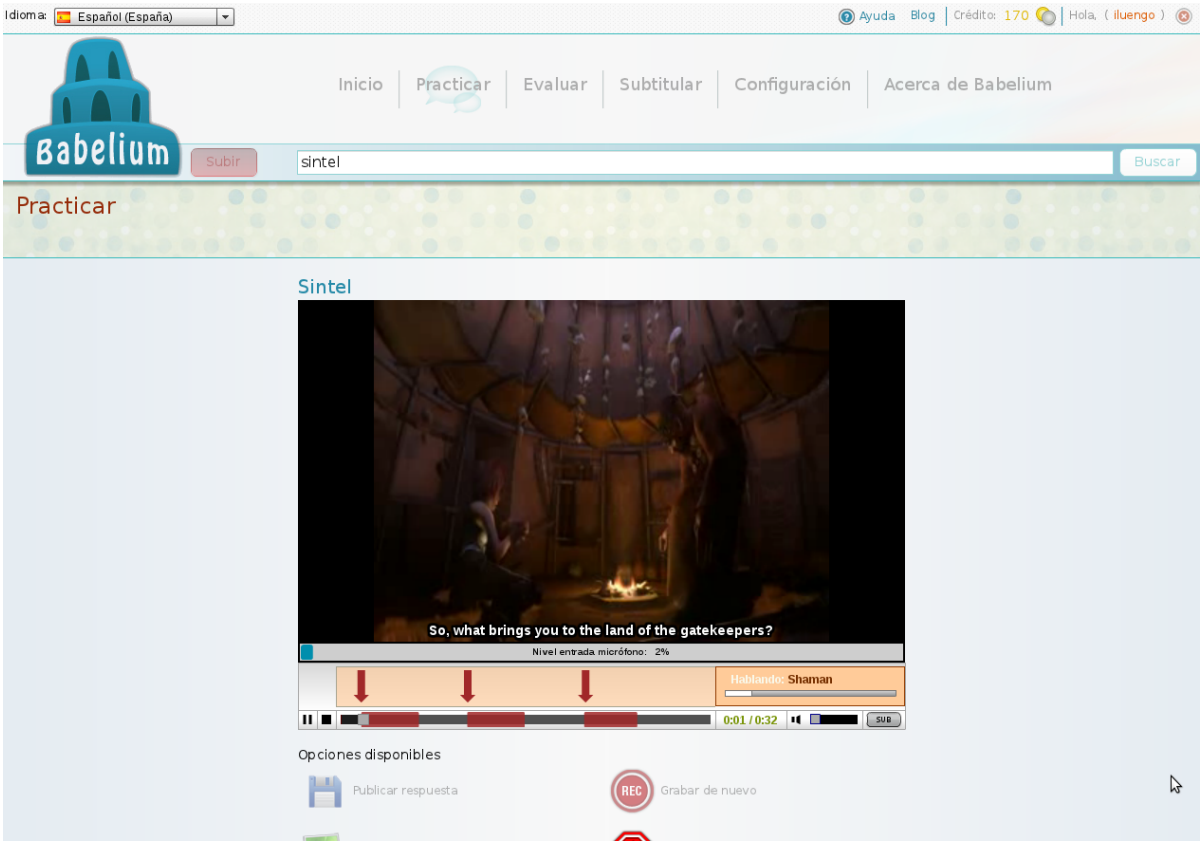


Figura 3.1: Vista rápida del módulo de grabación en Flex

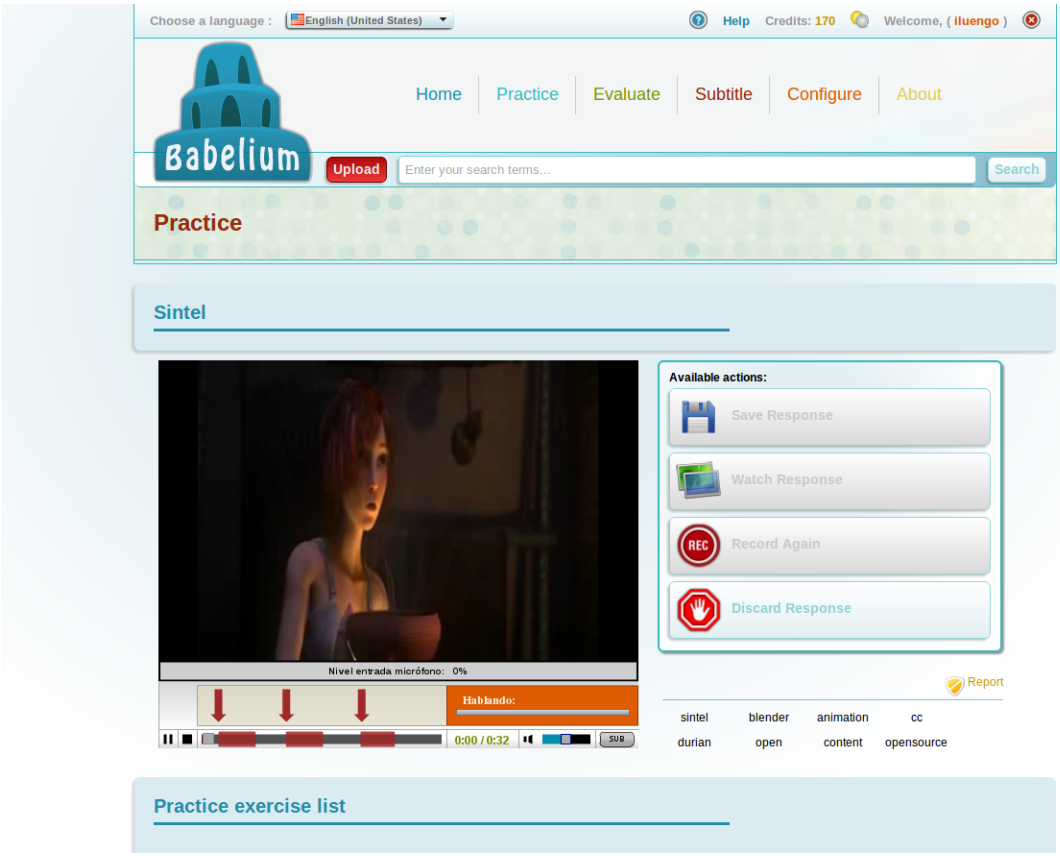


Figura 3.2: Vista rápida del módulo de grabación en HTML5

Capítulo 4

Control del Proyecto

Contents

El despliegue del proyecto se extiende desde principios de 2011, hasta finales de mayo del 2012. En ese periodo tuvieron lugar todas las fases del trabajo: la introducción al proyecto, la fase de aprendizaje, los primeros pasos y por su puesto, la planificación, análisis y desarrollo de los objetivos propuestos.

A lo largo de este capítulo se darán a conocer todas las buenas prácticas y tecnologías empleadas para llevar el trabajo al día y bien organizado. Se darán también detalles de la planificación llevada a cabo y de la metodología empleada. Al terminar este capítulo, el lector tendrá una idea mucho más clara de los aspectos que el proyecto contempla, y del esfuerzo requerido para llevar acabo todo el trabajo.

4.1. Alcance

El alcance del proyecto se definió pasados unos meses del planteamiento del mismo (como se verá más adelante en el apartado de planificación). Como se explicó anteriormente, los objetivos y requisitos iniciales eran demasiado genéricos: realizar una migración de software. Así pues, se definió un proceso de investigación, en el cual se investigó la factibilidad y una posible solución inicial, y empezando por ahí, siguiendo metodologías ágiles, se fue desarrollando por completo del proyecto hasta cumplir todos los objetivos y haber implementado todo lo apalabrado en el alcance.

Para reflejar el alcance del proyecto, la siguiente figura refleja el EDT (Estructura de Desglose de Trabajo) final:

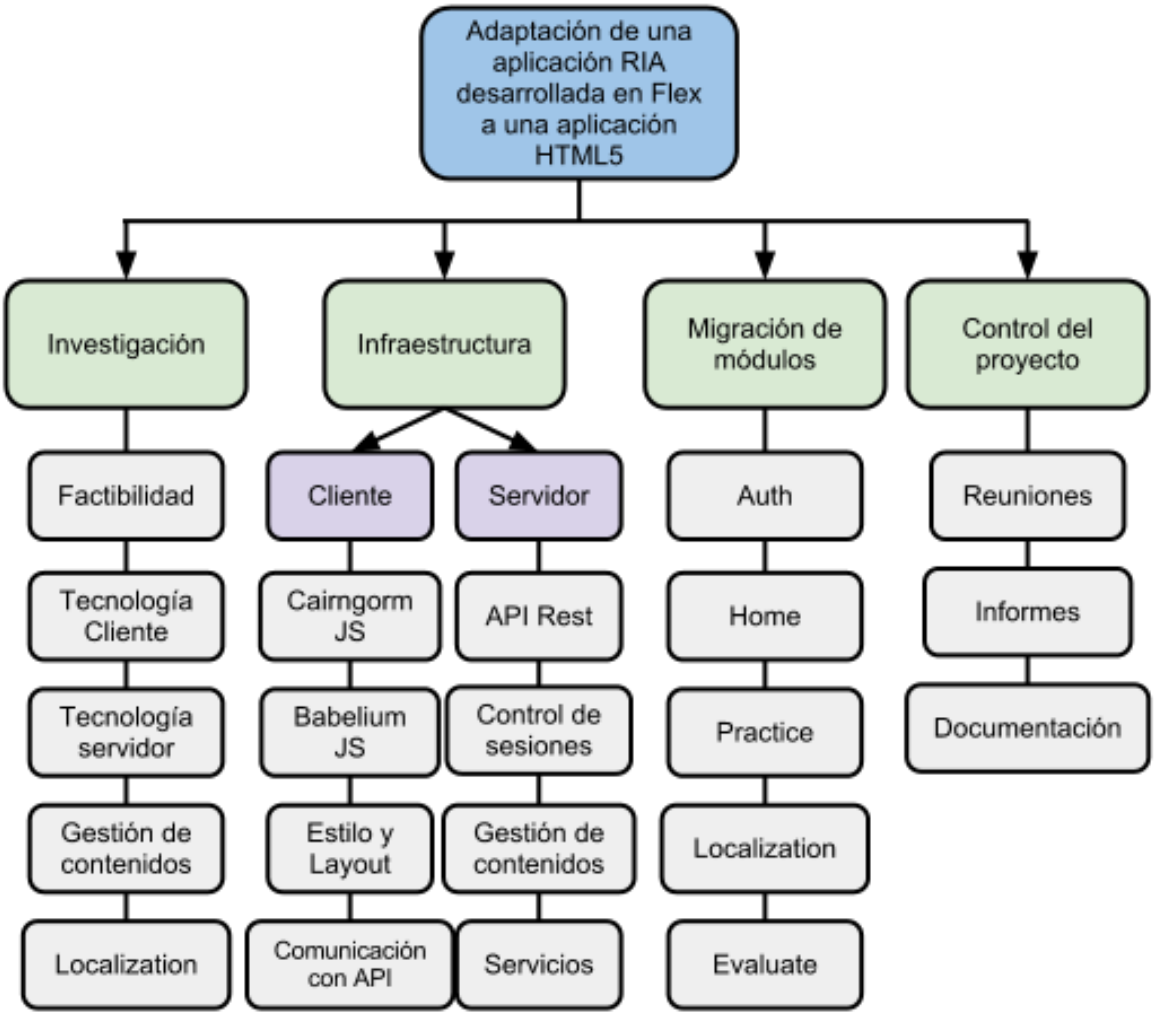


Figura 4.1: Gestión del alcance: EDT

4.2. Metodología

Dado que nos encontramos frente a un proyecto real, que necesita soluciones reales, se decidió seguir metodologías basadas en el *desarrollo ágil* de software. La principal ventaja de estas metodologías, es que no requieren llevar un costoso y estricto plan de acción, y además permiten hacer distinción entre los distintos estados en los que se encuentra cada parte concreta del trabajo.

Existen muchas otras metodologías de desarrollo, y muchas de ellas son derivadas del ciclo de vida en cascada. Estas metodologías requieren una rigurosa planificación y dividen el desarrollo del software principalmente en las siguientes iteraciones: identificación de requerimientos y casos de uso, análisis, diseño, implementación y pruebas; todas ellas en una estricta secuencia. Esta idea está muy bien como concepto teórico, o para aplicarla en proyectos pequeños donde gracias al modelado de software puede resumirse y planificarse el proyecto completo sin mucha demora. Sin embargo, requiere mucho tiempo de estudio, planificación y documentación antes de comenzar su desarrollo. Por ello, cuando hablamos de proyectos de gran envergadura, o de proyectos en constante desarrollo donde los requisitos y los objetivos cambian o crecen constantemente en el tiempo, vemos la necesidad de dejar de lado esta metodología y pasar a otra que nos ofrezca mayor flexibilidad.

El desarrollo ágil cambia el concepto de iteración que las otras metodologías proponen. Mientras en cascada es necesario completar todas las iteraciones para tener una versión funcional del software, haciéndolo de forma ágil convertimos cada iteración en una entrega de software. Se aparta el costoso trabajo de análisis y diseño global, partiendo la entrega de implementación final en distintos prototipos o funcionalidades. A cada prototipo se le asigna una iteración, que tendrá una duración entre una semana y un mes, al final del cual se entregará una beta sin errores de esa funcionalidad. Cada iteración tiene internamente su proceso de planificación y diseño, pero al hacerse individualmente y "sobre la marcha" para cada funcionalidad, permite abstraerse más de los requisitos, condiciones y objetivos, y centrarse en el desarrollo. Esto no quiere decir que la planificación y documentación no sea necesaria, si no que no tiene que ser tan estricta y es mucho más fácil hacer una *vuelta atrás*.

En el caso en el que los objetivos cambien, otras necesidades o funcionalidades aparezcan, o cambien los requisitos o condiciones del sistema, siguiendo la metodología de desarrollo en cascada, tendríamos que hacer una *vuelta atrás* y volver a hacer un análisis y diseño completo para reflejar esos cambios, así luego pasar a implementarlos. En este proyecto sin embargo, han ido aumentando varias veces dichos objetivos y requisitos, y se han añadido muchísimas funcionalidades inicialmente no reflejadas. Gracias a la metodología empleada, sólo ha sido necesario una reestructuración de una pequeña it-

eración, la creación de una nueva para adaptar los nuevos objetivos, o incluso ni eso ya que dichos objetivos estaban reflejados en iteraciones futuras y aún no comenzadas.

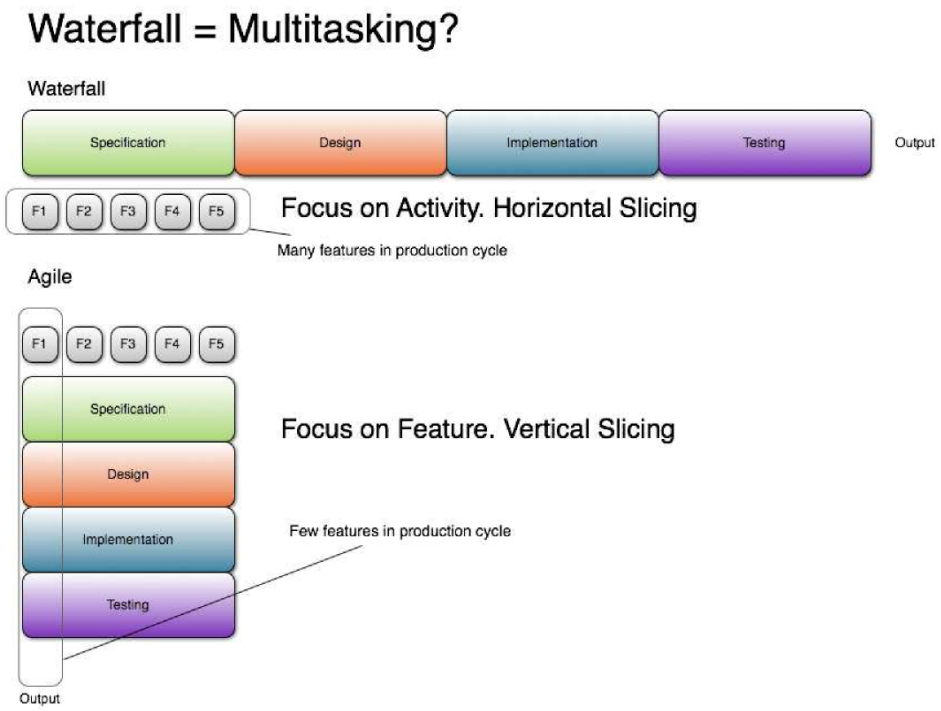


Figura 4.2: Desarrollo en cascada vs desarrollo ágil

En la figura 4.2 podemos ver la clara ventaja de una metodología frente a otra. El desarrollo ágil otorga, sin duda, una mayor productividad, calidad y eficiencia.

Dado que he sido el único desarrollador fijo participante en el proyecto, no vi necesario la utilización estricta de una metodología ágil, sin embargo, se utilizó un modelo muy similar al que propone la metodología Scrum (figura 4.3).

Se definieron distintas iteraciones de duración aproximada de 1 mes, exceptuando la inicial de investigación, que se realizó con calma a lo largo de varios meses antes de iniciar el proyecto. Al comienzo de cada iteración le dediqué unos días a la planificación de ésta, registrando en **Trello** la lista de tareas y subtareas completa que iban a ser necesarias para llevar la iteración a cavo. Durante la iteración, al final de cada día o tarea realizada se actualizaba en Trello, añadiendo comentarios en caso de que fueran precisos. Si se habían realizado cambios y/o implementaciones importantes, mandaba un email informando de los avances a Juanan (y a Inko en varias ocasiones). Al final de cada iteración dedicaba unos días a documentar todos los cambios y tareas realizadas a lo largo

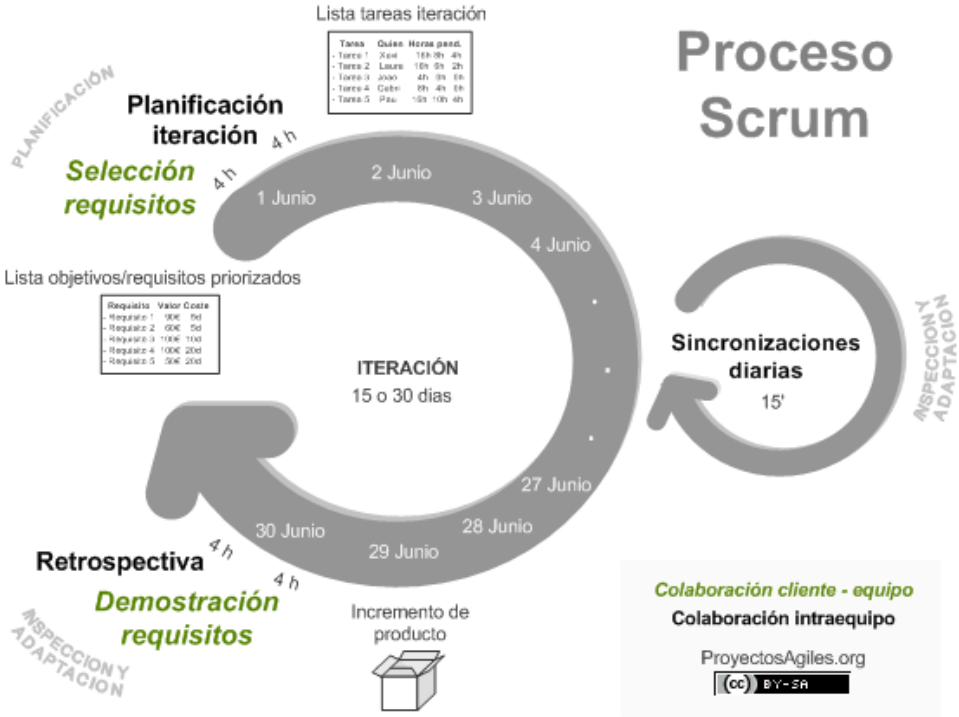


Figura 4.3: Metodologías ágiles: Scrum

de la misma. Finalmente, por email o mediante una reunión con Juanan, decidíamos el siguiente paso a realizar, y definíamos la siguiente iteración.

4.3. Planificación y organización

Objetivos complejos, requieren tareas complejas para llevarlos a cabo. Con el propósito de agilizar el desarrollo y mantener el trabajo claro y organizado, se hizo uso de distintas herramientas de planificación y organización de proyectos, que serán descritas en los siguientes apartados.

4.3.1. Planificación

La planificación del proyecto, en base a la metodología ágil utilizada, se realizo en base a iteraciones. Cada iteración correspondía a una funcionalidad o fase del proyecto, al final de la cual la aplicación avanzaba de forma incremental. Cada fin de iteración suponía una entrega de avances en el software, ya fueran funcionalidades adicionales, mejora de la infraestructura, etc..

Por ello, teniendo en cuenta el concepto global de Babelium Project, y la forma en la que funciona internamente, se dividió el proyecto completo en las siguientes iteraciones, todas ellas con su correspondiente sección dentro del capítulo 8.

Investigación: la primera fase fue en realidad una fase previa al inicio del proyecto. Le dediqué unos meses a la investigación sobre HTML5, su estado actual, y la posibilidad de migrar la aplicación completa, o al menos, gran parte de ella.

Prueba de concepto: iteración que pretende demostrar, mediante una infraestructura sencilla y un ejemplo rápido, que la migración de Flex a HTML5 es posible.

Infraestructura y gestión de sesiones: la iteración más larga y compleja, dedicada a desarrollar la infraestructura (los cimientos) de la aplicación, tanto en el cliente como en el servidor.

Protocolos de comunicación, navegación y API: iteración dedicada a mejorar y securizar los protocolos de comunicación entre cliente-servidor y la API.

Auth Module: implementación del login/logout y registro de usuarios, mediante confirmación de Email.

Home Module: implementación del módulo principal de la aplicación, que da la bienvenida al usuario y le facilita contenido interesante.

Practice Module: implementación del módulo practice, creado para que un usuario pueda grabar audio, y opcionalmente vídeo, de un ejercicio realizado, para luego ser evaluado por otros usuarios.

Localization: iteración que implementó un sistema para internacionalizar la web, y tenerla disponible en cualquier número de idiomas posibles.

Evaluation Module: implementación del módulo evaluación, creado para evaluar los ejercicios grabados por los usuarios en el módulo practice.

Como se ha mencionado en la sección anterior, relativa a la metodología, la planificación de cada una de las iteraciones se llevaba a cavo a lo largo de los primeros días de la misma, haciendo, si hicieran falta, cambios mínimos sobre la marcha. Dentro de cada iteración, se crearon "stories". Una *story* es una serie de tareas relacionadas entre sí dentro de una misma iteración. Y por último, al más bajo nivel, están todas y cada una de las tareas necesarias para completar una *story*. De esta forma, se consigue desglosar el concepto complejo de la funcionalidad completa de una iteración, en distintas *stories*, normalmente una por funcionalidad de la iteración, y dentro de cada *story*, una lista de tareas necesarias para que ésta se dé por terminada.

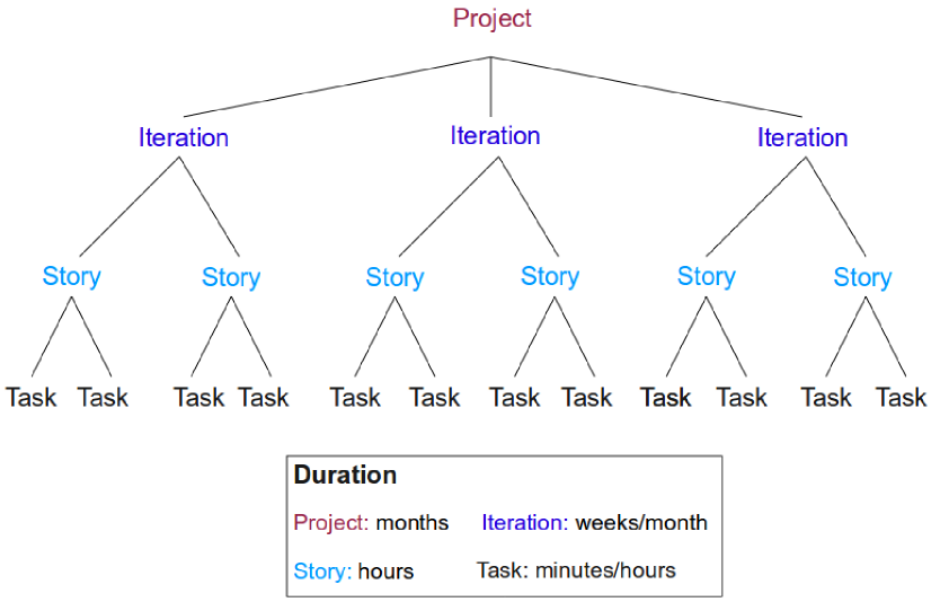


Figura 4.4: Árbol de planificación ágil

Para realizar la planificación del proyecto, se hizo uso de una herramienta online llamada **Trello** [8]. Trello es una página pensada para gestión de tareas, proyectos.. cualquier cosa que pueda gestionarse en base a notas. Es una plataforma online donde, una vez registrado, puedes crear cualquier cantidad de "proyectos". Es una web 100 %

colaborativa por lo que los proyectos pueden ser públicos o privados, y en cualquiera de ambos casos, se pueden enviar invitaciones a otros usuarios para que se unan al proyecto.

Una vez dentro del proyecto, pueden crearse tablones distintos, ya sea uno por proyecto, uno por iteración.. a gusto del usuario; y dentro de cada tablón, los usuarios que pertenezcan al proyecto pueden añadir entradas en forma de notas, que son actualizadas en tiempo real para el resto de usuarios del proyecto. Además, se puede crear cualquier número de categorías para las notas, y se pueden etiquetar las notas por colores. Por último, las notas pueden contener más cosas además de texto, por ejemplo: listas de tareas para dar por terminada la nota, comentarios de los desarrolladores, etc..

Antes de continuar explicando, observar en la figura 4.5 una imagen del tablón de trello donde uno o más usuarios han añadido notas.

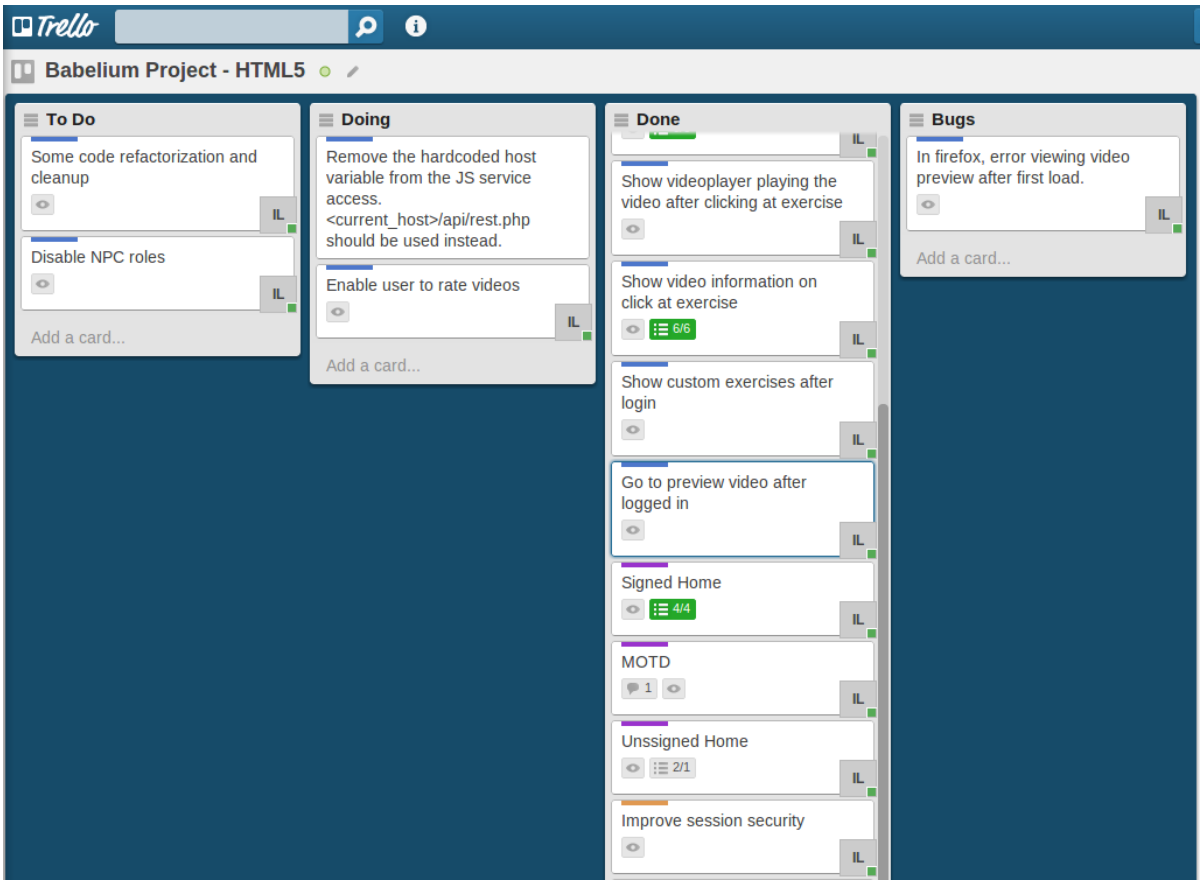


Figura 4.5: Planificación del proyecto: tablón principal de trello

Como se puede observar en la figura 4.5, para este proyecto se ha utilizado un sólo

tablón principal, que contiene todo el proyecto dividido en 4 categorías: ToDo (por hacer), Doing (haciendo), Done (hecho) y bugs (errores pendientes de corrección).

Además de las categorías, las notas pueden contener etiquetas. En este proyecto las etiquetas se utilizaron para definir la iteración a la que pertenecía cada nota, como puede observarse en la figura 4.6.

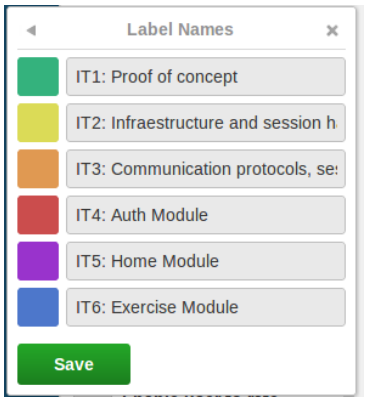


Figura 4.6: Planificación del proyecto: etiquetas en forma de iteraciones

Por último, como se ha mencionado antes, una nota puede ser algo más que un trozo de texto informativo, puede representar una tarea al completo. Trello hace esto posible mediante un seguimiento de actividad de las notas, posibilidad de añadir comentarios, y, sobre todo, posibilidad de añadir checklists para dar la tarea por terminada. En la figura 4.6 puede verse un fragmento del checklist y actividad de una tarea de la iteración de migración del módulo exercises.

La ventaja de usar esta herramienta, es que además de organizar la gestión de tareas y el alcance del proyecto, cualquier interesado en el proyecto podía rápidamente observar el estado del avance del mismo con sólo echar un vistazo al tablón de Trello. Es por ello que Juanan Pereira (supervisor) e Inko Perurena (desarrollador interesado) estaban como invitados en el tablón para observar el seguimiento del proyecto.

Gracias a la interfaz que ofrece Trello, se pudo planificad fácilmente cada una de las iteraciones, asignándole un color a la etiqueta para cada iteración, y planificando todas las tareas que se iban a realizar al comienzo de cada una de las iteraciones.

El primer paso, era siempre crear una etiqueta para la iteración, y posteriormente, se creaba una nota por cada story, es decir, por cada agrupación de tareas de un mismo interés, como podía ser: “crear la interfaz de usuario para el módulo home”, o “crear la lógica del módulo home’. Dentro de esa story, si contenía más de una tarea, se creaba un

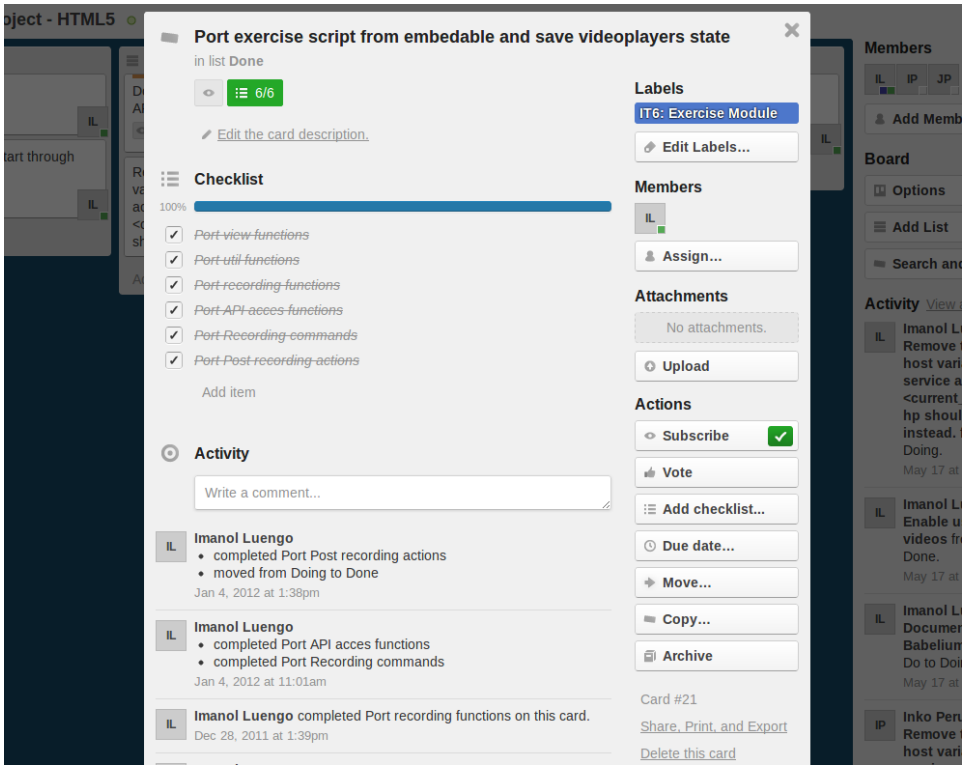


Figura 4.7: Planificación del proyecto: seguimiento de tareas

checklist con todas y cada una de ellas, desglosando de esta forma la iteración completa, en bloques de tareas simples y concretas.

Inicialmente, todas las stories (y con ellas, las tareas) de la iteración comenzaban en la categoría "ToDo", y, de la misma forma que se aplica la metodología GTD (siguiente apartado), se iban seleccionando una a una las stories que se iban a llevar a cabo, según la preferencia. Una vez se seleccionaba una story a realizar, se movía de la categoría "ToDo" a "Doing", y aquí se planificaba y se calculaba el tiempo esperado para completar todas y cada una de las tareas, teniendo así una cifra orientativa del esfuerzo que conllevaría realizarla.

Una vez la story era sacada de la categoría "ToDo" y pasaba a "Doing", era para llevarla a cabo en las próximas horas/días. Se realizaban commits periódicos sobre esa taréa, y se apuntaban las horas invertidas. Finalmente, al terminar de implementar o realizar todas las tareas necesarias para que la story quedara cerrada, ésta se movía a la categoría "Done" y se realizaba una breve comparación entre las horas invertidas y las esperadas, para así poder afinar mejor la planificación para futuras tareas.

En la figura 4.7 se puede observar una story con todas las tareas necesarias para completarse.

4.3.2. Organización

Se vio necesaria la utilización de sistemas de organización y gestión de tareas para poder llevar a cabo toda la carga acumulada, toda la carga de asignaturas, prácticas en empresa y proyecto de fin de carrera, hizo aumentar exponencialmente las tareas pendientes.

Tras estudiar los distintos sistemas de gestión de tareas, se escogió el sistema GTD (*Getting Things Done*), el cual ofrece una gran flexibilidad y no depende de un control estricto de la planificación.

David Allen creó el GTD, un método para gestionar eficientemente y sin estrés tu tiempo libre y la productividad de tus tareas. El principal fundamento de ésto, es la necesidad de liberar la mente de numerosas tareas y preocupaciones, y centrarse en la realización de dichas tareas. Para ello Allen se refiere a los bloc de notas, agendas, cuadernos, móviles, ordenadores... como *recipientes*. Cualquier cosa que sirva para anotar cosas, hará la función de recipiente. Cada vez que se nos ocurra alguna tarea que tengamos que llevar a cabo, tomamos uno de los dos siguientes caminos:

- *La regla de los 2 minutos:* si la tarea nos va a llevar menos de dos minutos, y podemos hacerla, la hacemos inmediatamente. Teniendo 2 minutos como número orientativo, siendo aproximadamente el tiempo que nos llevaría almacenarla en el recipiente y posponerla.
- En caso contrario, la almacenamos en un recipiente para sacarla de nuestra cabeza.

Al menos una vez por semana, se vacía cada recipiente. Se van sacando tareas y llevándolas a cabo. Una vez se saca una tarea, es para realizarla, nunca se devuelve al recipiente. Además, si un objetivo necesita de más de una tarea para darlo por terminado, se archiva como *proyecto*.

A día de hoy existen varias herramientas para gestionar esta metodología, que aunque no necesarias, pues se puede llevar a cabo con un papel y un boli, ayudan bastante. Las primeras y más conocidas son *Things* e *iGTD*, sin embargo son versiones de pago (y para mac). Como con todo lo demás en el proyecto, se hizo un estudio de alternativas libres, a ser posible gratuitas, encontrando así un gran número de aplicaciones que cumplían las expectativas, como por ejemplo: *Evernote*, *Ta-Da*, *Nirvana HQ*, *Get Things Gnome*, *Remember the Milk*, *Thinking Rock*, etc...

Tras una investigación sobre todas las posibilidades, me acabé decantando por Evernote. Evernote, además de soportar con creces las ideas principales de GTD, es mucho más que eso. Primero, porque es multiplataforma, tiene la aplicación principal vía web, pero también tiene aplicaciones para Android o iPhone.

Permite, desde cualquiera de sus aplicaciones y una vez identificado, registrar de forma rápida y sencilla una nota y, además, asignarle cualquier número de *tags* (etiquetas). De esta forma, se pueden agrupar varias notas o tareas mediante tags, formando así los denominados "proyectos".

Sin embargo, lo que me hizo decantarme por Evernote, y lo que realmente fue extremadamente útil y lo diferenció del resto de sistemas GTD, fue su extensión para Chrome. Evernote dispone de una extensión para Chrome que, estando navegando por las webs permite hacer click en la extensión y registrar automáticamente la web en forma de nota dentro de Evernote. Esto es muy útil, ya que podemos guardar en evernote, además de notas, páginas en "favoritos" etiquetadas mediante tags.

Por si esto fuera poco, la extensión de Evernote nos permite, además, guardar **solamente** el artículo que estamos leyendo en la página web, en lugar de su URL. Es decir, la extensión de Evernote crea una copia HTML del **contenido** del artículo que estás leyendo, y lo almacena dentro de una nota (en formato HTML, tal y como se ve en la web original).

En la figura 4.8 puede observarse la interfaz web de Evernote, con un listado de etiquetas, y un artículo capturado mediante la extensión de Chrome. Mientras que en la figura 4.9 se puede observar la extensión en funcionamiento.

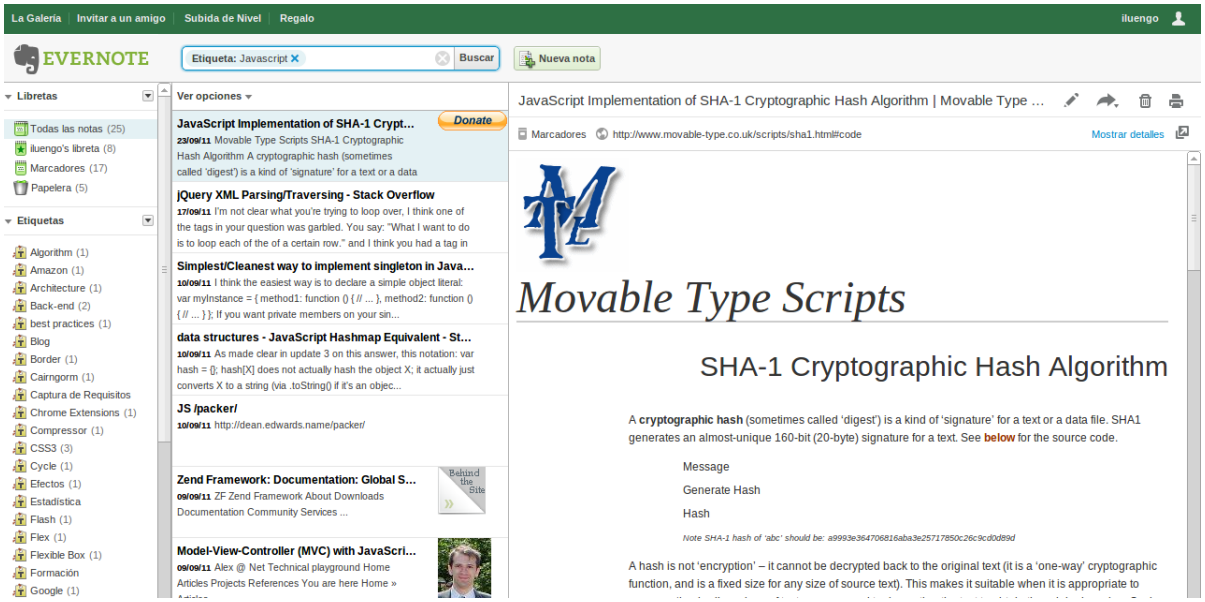


Figura 4.8: GTD: Evernote

Figura 4.9: GTD: Evernote extensión para Chrome

4.3.3. Análisis tiempo planificado vs tiempo real

En la lista de la página 23 se puede observar la lista de iteraciones que, juntas, han formado este proyecto de fin de carrera. Se ha seguido una metodología ágil para su planificación y gestión, por ello muchas de las iteraciones se han ido creando y gestionando sobre la marcha, sin dar lugar a grandes errores de planificación. Las iteraciones principales del proyecto han sido las siguientes:

El proyecto comenzó poco a poco a lo largo de enero de 2011, ya que sabía que iba a tener un par de años saturados de trabajo, decidí empezar el proyecto con antelación para ir alisando el terreno. Es por ello, que lo primero que realicé fue, a grosso modo y basándome en mi experiencia previa con el desarrollo de webs, crear un diagrama gantt de la duración e iteraciones (al nivel más abstracto) que pensaba que tendría la migración, para tener unas fechas orientativas:

	Nombre	Duración	Inicio	Terminado
1	⊟**PFCv2**	325 days?	**3/01/11 8:00**	**30/03/12 17:00**
2	Planificación	15 days?	28/03/11 7:00	15/04/11 17:00
3	Captura de requisitos	56 days?	3/01/11 8:00	21/03/11 17:00
4	Formación	303 days?	3/01/11 8:00	29/02/12 17:00
5	⊟**Desarrollo**	270 days?	**21/03/11 8:00**	**30/03/12 17:00**
6	Prototipo en HTML5	52 days?	21/03/11 8:00	31/05/11 17:00
7	Logica de negocio	22 days?	1/06/11 7:00	30/06/11 17:00
8	Comunicación entre capas	21 days?	1/07/11 7:00	29/07/11 17:00
9	Integración de funcionalidades básicas	43 days?	1/09/11 7:00	31/10/11 17:00
10	Integración de funcionalidades avanzadas	87 days?	1/10/11 7:00	31/01/12 17:00
11	Mejora de la interfaz	65 days?	2/01/12 8:00	30/03/12 17:00
12	⊟**Pruebas**	270 days?	**21/03/11 8:00**	**30/03/12 17:00**
13	Compatibiliad con Chrome y Firefox	270 days?	21/03/11 8:00	30/03/12 17:00
14	Compatibilidad con otros navegadores	65 days?	2/01/12 8:00	30/03/12 17:00

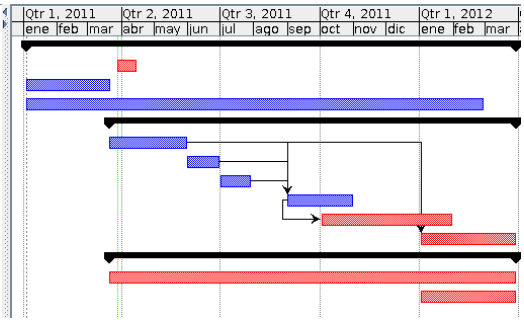
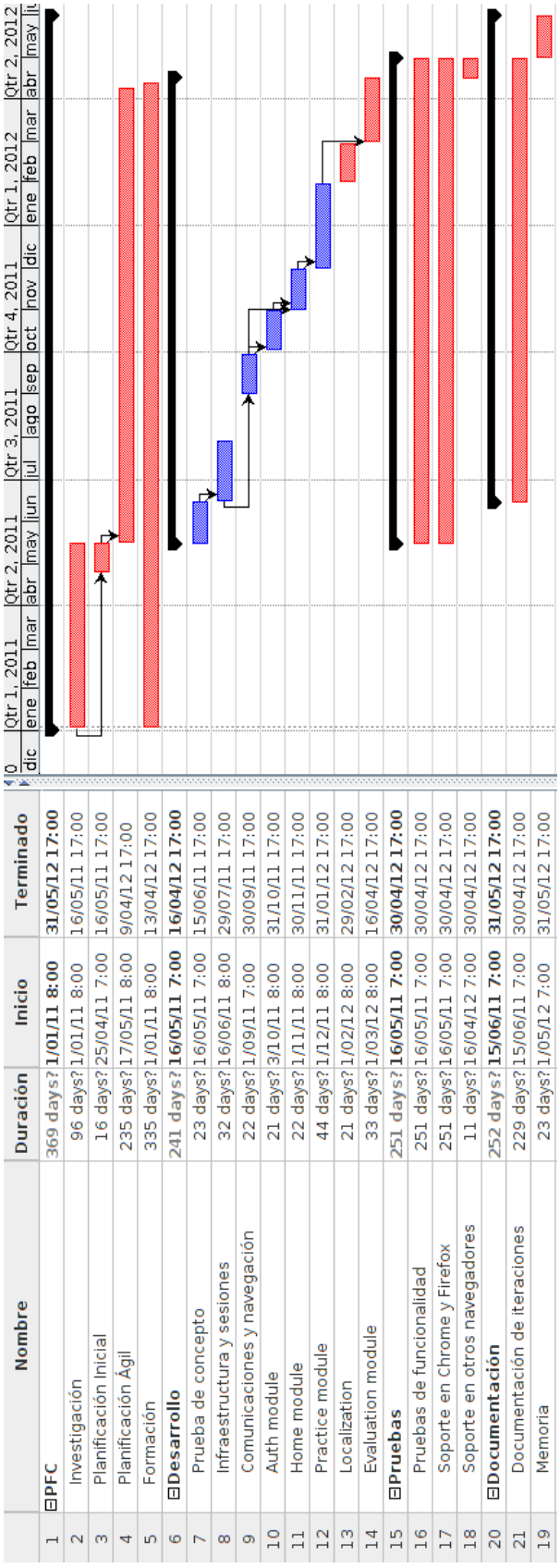


Figura 4.10: Diagrama gantt planificado antes del inicio del proyecto

Como ya he dicho, se hizo nada más pactar el proyecto, por lo que aún no tenía experiencia ni conocimientos suficientes en el tema, ni sabía las iteraciones que iba a tener el proyecto, sin embargo, si que sabía las partes que componen una web y las que iba a necesitar desarrollar. Como se puede ver en la figura 4.10, existe un proceso de captura de requisitos (o investigación), y después es cuando empieza la planificación del proyecto. Aunque he comentado que se ha seguido una metodología ágil de desarrollo, eso no implica que no haya que planificar las cosas con antelación. Es por ello que esa fase de planificación, una vez se le dedicaron aproximadamente 3 meses a la investigación de las tecnologías que forman HTML5 y las posibilidades de migración, así como las partes que iban a ser necesarias para llevar a cabo el proyecto, dio como resultado un nuevo diagrama de gantt, visible en la figura 4.11.



	Nombre	Duración	Inicio	Terminado
1	**PFC**	369 days?	**1/01/11 8:00**	**31/05/12 17:00**
2	Investigación	96 days?	1/01/11 8:00	16/05/11 17:00
3	Planificación Inicial	16 days?	25/04/11 7:00	16/05/11 17:00
4	Planificación Ágil	235 days?	17/05/11 8:00	9/04/12 17:00
5	Formación	335 days?	1/01/11 8:00	13/04/12 17:00
6	**Desarrollo**	241 days?	**16/05/11 7:00**	**16/04/12 17:00**
7	Prueba de concepto	23 days?	16/05/11 7:00	15/06/11 17:00
8	Infraestructura y sesiones	32 days?	16/06/11 8:00	29/07/11 17:00
9	Comunicaciones y navegación	22 days?	1/09/11 7:00	30/09/11 17:00
10	Auth module	21 days?	3/10/11 8:00	31/10/11 17:00
11	Home module	22 days?	1/11/11 8:00	30/11/11 17:00
12	Practice module	44 days?	1/12/11 8:00	31/01/12 17:00
13	Localization	21 days?	1/02/12 8:00	29/02/12 17:00
14	Evaluation module	33 days?	1/03/12 8:00	16/04/12 17:00
15	**Pruebas**	251 days?	**16/05/11 7:00**	**30/04/12 17:00**
16	Pruebas de funcionalidad	251 days?	16/05/11 7:00	30/04/12 17:00
17	Soporte en Chrome y Firefox	251 days?	16/05/11 7:00	30/04/12 17:00
18	Soporte en otros navegadores	11 days?	16/04/12 7:00	30/04/12 17:00
20	**Documentación**	252 days?	**15/06/11 7:00**	**31/05/12 17:00**
21	Documentación de iteraciones	229 days?	15/06/11 7:00	30/04/12 17:00
19	Memoria	23 days?	1/05/12 7:00	31/05/12 17:00

Figura 4.11: Diagrama gantt planificado

Como se puede observar en la imagen, hay unos meses de investigación y formación previos a la planificación del proyecto (de donde salió el diagrama gantt en cuestión) y las iteraciones que iban a ser necesarias para completar el proyecto. Después se ve una planificación por iteración, que dura todo el proyecto, aunque sólo afecta el inicio y final de cada iteración, no al tiempo intermedio; además de un periodo de formación, que abarca todas las iteraciones.

Después se observan las iteraciones principales del proyecto, dentro de "Desarrollo". Este apartado de desarrollo de la aplicación en HTML5, se extiende desde mayo hasta abril, casi un año, anticipando 1 mes de vacaciones en verano (no sabía si iba a ser julio o agosto, pero en el diagrama aparece como agosto).

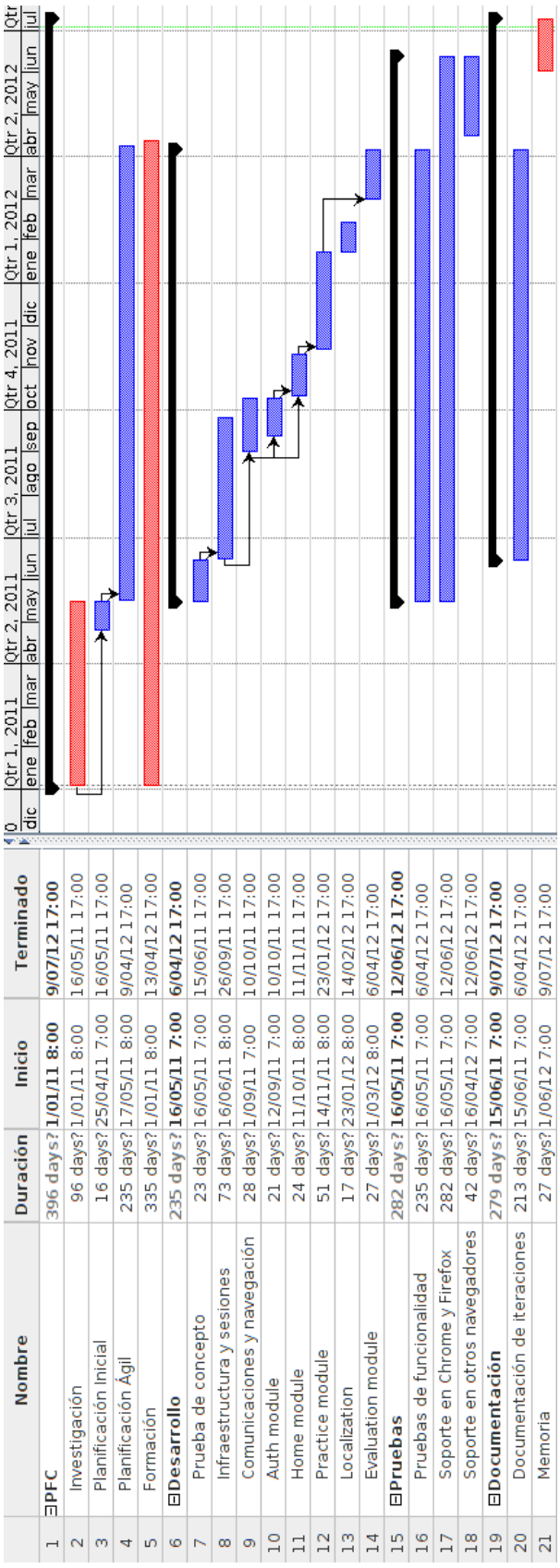
Después, se puede observar como hay unos procesos de pruebas, que acompañan al desarrollo y se va comprobando el correcto funcionamiento (especialmente al final de cada iteración).

Y por último, la documentación, con 2 procesos, uno que se realizó a lo largo del proyecto, documentando en Google Docs las tareas realizadas al final de cada iteración, y otro con la memoria, la redacción de este documento.

No hay muchas diferencias entre lo planeado y lo real. Las difrencias más remarcables, se encuentran en:

- Las iteraciones de **Infraestructura**, **Comunicaciones** y **Auth Module** se realizaron en paralelo, en lugar de en serie, ya que estaban relacionadas entre sí y sentaban las bases del resto de iteraciones. Además, se consiguió que, con una cantidad de esfurzo similar, se adelantara la fecha de fin de las iteraciones.
- El resto de iteraciones siguen su curso previsto, con algunos adelantos en las entregas debido al paralelismo mencionado en el punto anterior.
- Se añadió un periodo de descanso al comienzo de febrero, debido a la necesidad de unas vacaciones por las prácticas en empresa y los exámenes de febrero.
- Se terminó de desarrollar el proyecto antes de lo previsto, a comienzos de abril para poder estar libre para los exámenes y asignaturas del segundo cuatrimestre.
- Por contra, la tarea de compatibilidad la aplicación con el resto de navegadores (además de chrome y firefox) llevó más tiempo de lo esperado (también debido a la carga de trabajos y exámenes) y se extendió hasta comienzos de junio.
- Debido al último punto, la redacción de la memoria se realizó entre junio y julio.

Todos los puntos mencionados anteriormente pueden observarse reflejados en la figura 4.12



	Nombre	Duración	Inicio	Terminado
1	⊟**PFC**	396 days?	**1/01/11 8:00**	**9/07/12 17:00**
2	Investigación	96 days?	1/01/11 8:00	16/05/11 17:00
3	Planificación Inicial	16 days?	25/04/11 7:00	16/05/11 17:00
4	Planificación Ágil	235 days?	17/05/11 8:00	9/04/12 17:00
5	Formación	335 days?	1/01/11 8:00	13/04/12 17:00
6	⊟**Desarrollo**	235 days?	**16/05/11 7:00**	**6/04/12 17:00**
7	Prueba de concepto	23 days?	16/05/11 7:00	15/06/11 17:00
8	Infraestructura y sesiones	73 days?	16/06/11 8:00	26/09/11 17:00
9	Comunicaciones y navegación	28 days?	1/09/11 7:00	10/10/11 17:00
10	Auth module	21 days?	12/09/11 7:00	10/10/11 17:00
11	Home module	24 days?	11/10/11 8:00	11/11/11 17:00
12	Practice module	51 days?	14/11/11 8:00	23/01/12 17:00
13	Localization	17 days?	23/01/12 8:00	14/02/12 17:00
14	Evaluation module	27 days?	1/03/12 8:00	6/04/12 17:00
15	⊟**Pruebas**	282 days?	**16/05/11 7:00**	**12/06/12 17:00**
16	Pruebas de funcionalidad	235 days?	16/05/11 7:00	6/04/12 17:00
17	Soporte en Chrome y Firefox	282 days?	16/05/11 7:00	12/06/12 17:00
18	Soporte en otros navegadores	42 days?	16/04/12 7:00	12/06/12 17:00
19	⊟**Documentación**	279 days?	**15/06/11 7:00**	**9/07/12 17:00**
20	Documentación de iteraciones	213 days?	15/06/11 7:00	6/04/12 17:00
21	Memoria	27 days?	1/06/12 7:00	9/07/12 17:00

Figura 4.12: Diagrama gantt real

Desglose por iteraciones

La planificación inicial, reflejada en la figura 4.11 se realizó estimando una inversión media de 1 hora diaria en la fase previa al proyecto, investigación y formación, y 3 horas diarias en el desarrollo del PFC, con lo que tendríamos:

$$96 días * \frac{1 hora}{día} + 273 días * \frac{3 horas}{día} = \textbf{915 horas}$$

El proyecto se planificó para una duración de algo más de año y medio, 369 días aproximadamente sin contar festivos y vacaciones, lo que da un total de 915 horas que esperaban dedicarse al proyecto. A continuación se muestra un resumen de la diferencia entre el tiempo planificado y el real, y un desglose de tiempo medio ocupado por tipo de tarea:

	Planificado	**Real**
Total	915 horas (369 días)	985 horas (396 días)

Cuadro 4.1: Tiempo planificado vs real

Tarea	**Tiempo**	**Tiempo medio**
Total	**985 horas**	**2.48h/día**
Procesos tácticos (planificación, formación, documentación...)	270 horas	0.68h/día
Procesos operativos (diseño, implementación, redacción...)	715 horas	1.80h/día

Cuadro 4.2: Esfuerzo real desglosado por tipo de tarea

Ahora un resumen del esfuerzo real en las iteraciones desglosadas por tipo de tareas:

Investigación	
Periodo: 01/01/2011 - 16/05/2011	±5 meses
Dedicación media (96 días):	1.45h/día
Tarea	**Tiempo**
Total	±140h
Investigación	±30h
Formación	±60h
Documentación	±10h
Pruebas	±40h

Cuadro 4.3: Esfuerzo real: Investigación

Prueba de concepto	
Periodo: 16/05/2011 - 15/06/2011	±1mes
Dedicación media (23 días):	1.95h/día
Tarea	**Tiempo**
Total	±45h
Formación	±10h
Planificación	±5h
Documentación	±5h
Diseño	±5h
Implementación	±20h

Cuadro 4.4: Esfuerzo real: Prueba de concepto

Infraestructura y gestión de sesiones	
Periodo: 16/06/2011 - 26/09/2011	±3meses
Dedicación media (73 días):	2.4h/día
Tarea	**Tiempo**
Total	±175h
Formación	±30h
Planificación	±5h
Documentación	±10h
Diseño	±10h
Implementación	±120h

Cuadro 4.5: Esfuerzo real: Infraestructura y gestión de sesiones

Comunicación entre capas y navegación	
Periodo: 01/09/2011 - 10/10/2011	±1 mes y medio
Dedicación media (28 días):	1.25h/día
Tarea	**Tiempo**
Total	±35h
Formación	±5h
Planificación	±1h
Documentación	±3h
Diseño	±5h
Implementación	±20h

Cuadro 4.6: Esfuerzo real: Comunicación entre capas y navegación

Auth Module	
Periodo: 12/09/2011 - 10/10/2011	±1 mes
Dedicación media (±21 días):	1.6h/día
Tarea	**Tiempo**
Total	±35h
Formación	±5h
Planificación	±2h
Documentación	±5h
Diseño	±1h
Implementación	±20h

Cuadro 4.7: Esfuerzo real: Auth Module

Home Module	
Periodo: 11/11/2011 - 11/11/2011	±1 mes
Dedicación media (24 días):	3.75h/día
Tarea	**Tiempo**
Total	±90h
Formación	±10h
Planificación	±5h
Documentación	±5h
Diseño	±10h
Implementación	±60h

Cuadro 4.8: Esfuerzo real: Home Module

Practice Module	
Periodo: 14/11/2011 - 23/01/2012	±2 meses
Dedicación media (51 días):	2.35h/día
Tarea	**Tiempo**
Total	±120h
Formación	±5h
Planificación	±5h
Documentación	±2h
Diseño	±8h
Implementación	±100h

Cuadro 4.9: Esfuerzo real: Practice Module

Localization	
Periodo: 23/01/2012 - 14/02/2012	±3 semanas
Dedicación media (17 días):	2.95h/día
Tarea	**Tiempo**
Total	±50h
Formación	±10h
Planificación	±1h
Documentación	±5h
Diseño	±2h
Implementación	±30h

Cuadro 4.10: Esfuerzo real: Localization

Evaluation Module	
Periodo: 01/03/2012 - 06/04/2012	±1 mes
Dedicación media (27 días):	3.8h/día
Tarea	**Tiempo**
Total	±105h
Formación	±5h
Planificación	±5h
Documentación	±10h
Diseño	±5h
Implementación	±80h

Cuadro 4.11: Esfuerzo real: Evaluation Module

Pruebas	
Periodo: 16/05/2011 - 12/06/2012	1 año
Dedicación media (282 días):	±0.17h/día
Tarea	**Tiempo**
Total	±50h
Pruebas de funcionalidad	±10h
Soporte para Chrome y Firefox	±10h
Soporte para el resto de navegadores	±30h

Cuadro 4.12: Esfuerzo real: Pruebas

Redacción de la memoria	
Periodo: 1/06/2012 - 09/07/2012	1.5 meses
Dedicación media (27 días):	5.1h/día
Tarea	**Tiempo**
Total	±140h
Formación	±10h
Redacción y maquetación	±130h

Cuadro 4.13: Esfuerzo real: Redacción de la memoria

4.4. Control

La gestión interna del proyecto se divide en apartados, los cuales utilizan sus respectivas herramientas. Por una parte está el *control de la información*, donde se encuentran los métodos y herramientas para gestionar toda la información relevante al proyecto; y por otra parte está el *control de versiones*, que incluye herramientas para gestionar el código fuente de la aplicación.

4.4.1. Control de la información

Smartphone Android

Mediante un smartphone con Android, con conexión directa a internet, y por ende a GMail, podía estar conectado a Babelium Project, leer los avances, y debatir en el grupo y en los emails con alta disponibilidad. Además de la posibilidad de añadir tareas a mi gestor GTD (Evernote o Catch) desde cualquier lugar y en cualquier momento.

Catch

Del gestor GTD Evernote ya hemos hablado en la sección de planificación, pues es el gestor que se ha usado principalmente para guardar las notas del proyecto. Sin embargo, Catch ha sido otra aplicación para Android de mucha utilidad a lo largo del proyecto. Es una aplicación realmente simple, que en 2 clicks permite guardar notas rápidas. Aunque Evernote es mucho más potente, Catch funciona sin internet y es mucho más rápida y simple. Es por ello que en ocasiones puntuales, donde había prisa o falta de conexión, se ha hecho uso de Catch para guardar notas.

Google Groups

Aunque no fue un método de comunicación relevante (concretamente) a este proyecto de fin de carrera, si que lo fue para estar en contacto con el resto de desarrolladores trabajando en otros proyetos dentro de Babelium Project. Se creó el grupo babeliumprojectgooglegroups.com con la finalidad de mantenernos a todos en contacto, estar al tanto de los avances que se hacían en el proyecto a nivel global, o incluso para publicar noticias que podían ser de gran interés para Babelium Project.

GMail

Método principal de comunicación del proyecto, mediante el cual se han tenido largas conversaciones de emails, para resolver dudas, informar de avances, o debatir sobre posibles implementaciones o conceptos del proyecto.

GMail es una potentísima herramienta de gestión de correo electrónico. Muchos de los debates, discusiones y dudas se han resuelto por este método. GMail dispone de un sistema de búsqueda de mails, además de un sistema de *etiquetas* para catalogar los emails que llegan. Gracias a esto puedes tener una bandeja de entrada organizada y no perder el tiempo buscando entre 50 emails el que te interesa. Para facilitar aún más la cosa y automatizar la catalogación, GMail dispone de *filtros*. Los filtros son casos especiales bajo los cuales sucede una etiquetación automática.

Por ejemplo, relacionados con el proyecto, se creó la etiqueta "PFC", que almacenaba todos los mails enviados al grupo Babelium Project y los recibidos de forma directa por los miembros del mismo (Juanan e Inko).

Google Docs

Una de las herramientas de google que ofrece la posibilidad de crear (o subir) documentos docs o draw (hay más, pero estos son los que se han usado en el proyecto). Para cada iteración se creó un documento con requisitos previstos, estimaciones de duración y problemas posibles, así como detalles de la solución para cada iteración al final de la misma. Así mismo, se utilizó draw para crear los esquemas y diagramas (que también están disponibles a lo largo de esta memoria).

Además, una de las ventajas de utilizar Google Docs frente a redactar lo mismo en Word, es que Google Docs permite compartir documentos online con otros usuarios, y permite que más de un usuario pueda editar el contenido.

Dropbox

Aunque no ha sido un medio de intercambio de comunicación extremadamente usado, en ocasiones puntuales se han compartido archivos y carpetas mediante esta herramienta.

Blog

Aunque sólo usado en la fase de investigación, creé un blog para ir comentando y documentando las herramientas interesantes que iba encontrando para el desarrollo de HTML5 [9].

http://html5.babeliumproject.com

4.4.2. Control de versiones

Concepto muy importante cuando más de un desarrollador trabaja sobre el mismo código. Es inviable que si más de una persona trabajan sobre la misma base, lo hagan individual e independientemente. Por ello existe el concepto *servidor remoto*, un ordenador en internet al que todos los desarrolladores tienen acceso, y en este caso, que alberga en su disco duro información sobre la versión más actualizada del proyecto para que todos los miembros puedan consultarla y trabajar sobre ella. El control de versiones permite a los programadores trabajar en local[1] y luego guardar los cambios en un servidor remoto para que el resto de usuarios vean reflejados los cambios. Inicialmente se usaban protocolos de transferencia de ficheros, como por ejemplo SCP (*Secure Copy*) o FTP, para actualizar los ficheros en el servidor. La ventaja de usar una herramienta de control de versiones, frente a una FTP (*File Transfer Protocol*), por ejemplo, es que con la segunda se sube una copia actualizada de un fichero al servidor y se reemplaza por la existente. Sin embargo, si otro usuario simultáneamente estaba trabajando en el mismo fichero, y sube su fichero sin haber actualizado tus cambios, puede darse el caso (más habitualmente de lo que uno puede llegar a pensar) de que la segunda subida pise y deshaga los cambios realizados por la primera.

Para solventar esto se crearon los sistemas de control de versiones, que internamente llevan un control de versiones distintas de cada fichero. Así, cuando un usuario realiza cambios y quiera subirlos al servidor, sólo se enviarán los cambios realizados, no todo el fichero. Así conseguimos que dos usuarios puedan modificar el mismo fichero simultáneamente sin solapar el trabajo el uno al otro. Y en caso de haber conflictos, pues dos desarrolladores pueden editar al mismo tiempo el mismo trozo de código, avisan del error y ofrecen al desarrollador distintas posibilidades para solucionarlo.

Babelium Project estaba hospedado en Google Code, utilizando Mercurial como gestor para el control de versiones. Una vez creado el repositorio[2] en el servidor remoto, los clientes (desarrolladores) tan sólo tenían que descargarlo a su ordenador y

[1] Una versión del programa instalada en su propio ordenador.

[2] Carpeta en el que contiene el código del proyecto y la información de todas sus versiones

empezar a trabajar ahí. Periódicamente cada usuario iba actualizando los cambios que iba realizando, así como descargando y actualizando su copia local con los cambios que el resto de programadores hacían.

Branches

Pese a la gran ventaja que ofrece el control de versiones, a veces sigue siendo insuficiente. En nuestro caso, Babelium Project tenía alojado en el repositorio de Google Code la versión de Flex, y este proyecto, no tiene nada que ver con el código de la versión en Flex. Por ello, para trabajar sobre el mismo repositorio sin pisar el código de Flex, se hizo uso de las "ramas" (traducción literal del término técnico *branches* del inglés), una funcionalidad de los gestores de control de versiones.

Con las ramas, el proyecto pasa a tener varios hilos de desarrollo. Cuando se crea una rama, internamente se copia a otro destino una versión actualizada del proyecto. Y a partir de ahí, el desarrollador trabajará sobre dicha rama, añadiendo cambios a la rama sin afectar al desarrollo estable del proyecto. Una vez se haya terminado el trabajo crítico, el desarrollador encargado de la rama, pasa los cambios realizados a la versión estable, para que se reflejen allí también los cambios.

Para este proyecto, se creó una rama aparte llamada **html5**, que contiene únicamente el código del proyecto en HTML5, y que sólo comparte con la rama Flex la API y los servicios. Además, se creo otra rama, llamada **embeddable-videoplayer** con el único propósito de independizar el reproductor de vídeo del resto de la aplicación en Flex.

Aunque el uso habitual de las branches, es para crear una copia del proyecto y editar código crítico sin romper la versión estable; para este proyecto se usaron las branches para desarrollar nuevas herramientas para Babelium Project (la versión HTML5 y el reproductor de vídeo independizado) dentro del mismo repositorio de código.

Merges

Otra potente funcionalidad del control de versiones. Como comentábamos anteriormente, a veces se ve necesaria la creación de ramas para gestionar mejor el avance del proyecto. Por otro lado, puede pasar un largo periodo entre la creación de una rama, y la resolución de la misma. En estos casos, al finalizar la rama hay que portar todos los cambios de la rama a la versión estable, sin embargo, puede que la versión estable haya avanzado muchísimo desde entonces, e incluso que se hayan modificado mismos trozos de código que en la rama. Esto supondría una costosa operación de traspaso de

modificaciones. En algunos casos supondría, incluso, más tiempo para la actualización de la versión estable del que se ha empleado en la realización de los cambios.

Por ello, existe la automatización del traslado de los cambios mediante *merges*. Al realizar un merge, se traspasan automáticamente todos los cambios realizados de un sitio a otro, y en caso de que en el destino se hayan modificado las mismas partes que en el origen, esas partes se marcan como *conflictos*. Una vez terminado el merge, hay 3 posibilidades de corrección de conflictos.

- Corregirlos con la versión local del archivo: se usa el archivo que tiene el desarrollador en local para corregir los conflictos.
- Corregir con la versión remota del archivo: se usar el archivo del repositorio remoto para corregir los conflictos.
- Corregir manualmente: permite abrir el fichero que contiene marcados los conflictos, y los fragmentos de código de las 2 versiones (local y remota) para elegir alguno de los 2, o incluso reemplazar el conflicto con nuevo código ajeno a las 2 versiones. Este método suele ser el más usado.

A lo largo de este proyecto, se han utilizado los merges para actualizar la API y los servicios de la versión HTML5, con las últimas versiones que Inko Perurena creaba y actualizaba para la versión de Flex (se comentará más adelante la estructura y funcionamiento de la aplicación).

4.5. Riesgos y factibilidad

Como todo proyecto complejo, éste también tenía sus riesgos tanto por causas personales como por problemas de factibilidad o recursos tecnológicos. Es por ello que se han dividido los riesgos en dos subapartados, los comunes a todos o casi todos los proyectos informáticos, y los riesgos concretos de este proyecto.

4.5.1. Riesgos comunes

A continuación unas tablas que los resumen:

Riesgo:	**Conocimientos tecnológicos insuficientes**
Impacto:	Crítico
Descripción:	Cuando el desarrollador no tiene conocimientos suficientes de las tecnologías a utilizar para realizar su tarea.
Consecuencias:	El desarrollador no puede llevar a cabo los objetivos encomendados.
Solución:	Se dedicará un periodo de aproximadamente 4-5 meses, previos al inicio del proyecto, para investigar sobre las tecnologías: sus compatibilidades, las carencias de HTML5; así como dominar las tecnologías que forman HTML5.

Cuadro 4.14: Riesgo: Conocimientos tecnológicos insuficientes

Riesgo:	**Cambio de requisitos**
Impacto:	Crítico
Descripción:	Se da cuando el cliente del proyecto cambia los requisitos y/o objetivos del mismo durante su transcurso.
Consecuencias:	Posible retraso en las entregas debido a una replanificación que contemple los nuevos requisitos.
Solución:	Se utilizará una metodología ágil de desarrollo, que entre otras cosas, sirven para solucionar este tipo de problemas, pues la planificación es individual por cada funcionalidad o iteración, y no global para todo el proyecto, facilitando así cambios imprevistos en cualquiera de los apartados.

Cuadro 4.15: Riesgo: Cambio de requisitos

Riesgo:	**Errores de planificación**
Impacto:	Crítico
Descripción:	Al enfrentarme a proyecto de tal envergadura, es posible que puedan darse fallos de planificación por sobrevalorar (o infravalorar) la complejidad de alguna tarea.
Consecuencias:	Posible retraso en las entregas.
Solución:	Al igual que el riesgo anterior, se utilizará una metodología ágil que permitirá desglosar cada iteración en tareas concretas facilitando la valoración de su complejidad y la estimación aproximada de su duración.

Cuadro 4.16: Riesgo: Errores de planificación

Riesgo:	**Perdida total o parcial del código fuente**
Impacto:	Muy crítico
Descripción:	Factores como errores informáticos, humanos o robo del hardware pueden implicar perdidas en la implementación de la solución.
Consecuencias:	Perdida de parte de las horas invertidas en el proyecto.
Solución:	Utilización de Google Code para almacenar todo el código fuente en un repositorio Mercurial, teniendo así una copia de seguridad en un equipo remoto.

Cuadro 4.17: Riesgo: Perdida total o parcial del código fuente

Riesgo:	**Perdida total o parcial de la documentación**
Impacto:	Muy crítico
Descripción:	Los mismos factores mencionados en el riesgo anterior pueden afectar a la perdida de la documentación.
Consecuencias:	Perdida de horas invertidas en la documentación del proyecto
Solución:	Se irá albergando en Google Docs toda la documentación de cada iteración del proyecto. Para ir construyendo la memoria del proyecto de manera incremental y a su vez, teniendo una versión online siempre disponible. Alternativamente, Google Docs ofrece la posibilidad de descargar el contenido de un documento en formato ODT al ordenador, para tener una copia de seguridad local. Además, para evitar la perdida local de documentación durante la redacción de la memoria, se realizaron backups incrementales cada hora, así como totales diarios en un disco duro externo y en otra partición. Más abajo se muestra el trozo de código de un script que añadido al crontab, realiza copias incrementales cada hora de los archivos de la memoria.

Cuadro 4.18: Riesgo: Perdida total o parcial de la documentación

Script de copias incrementales comentado en el riesgo anterior:

```
1 echo "Creando backup: $(date "+%F alas %H") ..." >> /datos/memoria/backup.log
2 find /home/iluengo/memoria -type f -mtime -0.041
3     > /datos/memoria/filelist
4     2>> /datos/memoria/backup.log
5 tar czfT /datos/memoria/backup-$(date "+%F-%H").tgz /datos/memoria/filelist
6     2>> /datos/memoria/backup.log
7 echo "Terminado backup..." >> /datos/memoria/backup.log
```

4.5.2. Riesgos específicos

Riesgo:	**Objetivos del proyecto abstractos**
Impacto:	Crítico
Descripción:	Dado que el objetivo principal del proyecto es demasiado genérico "migración de software" puede darse el caso de tener problemas de planificación a la hora de encaminar bien el desarrollo del proyecto.
Consecuencias:	Conllevaría una implementación ineficaz y fuera de plazos de los objetivos.
Solución:	Utilizando los meses de investigación para, además de formarme, dividir el proyecto en iteraciones más concretas; utilizar metodología ágil de desarrollo y planificar y desglosar los objetivos de cada iteración al comienzo de esta.

Cuadro 4.19: Riesgo: Objetivos del proyecto abstractos

Riesgo:	**Soporte HTML5**
Impacto:	Muy Crítico
Descripción:	Ocurrirá si se descubre que HTML5 no está lo suficientemente avanzado e implementado en todos los navegadores como para soportar todas las funcionalidades de la versión de Flex.
Consecuencias:	Imposibilidad de migrar algunas de las funcionalidades de Flex a HTML5
Solución:	Este es el principal riesgo del proyecto, que casi con el 100 % de probabilidades que ocurrirá, pues HTML5 aún es un prototipo, y el que más tiempo va a llevar solventarlo. La mayoría de las funcionalidades de Babelium en Flex (pues no deja de ser una web), son factibles en HTML5; implementables de forma más o menos difícil, pero factibles. Sin embargo, el caso que principalmente nos preocupa es la captura de vídeo y audio, y en caso de que HTML5 no llegue a soportarla, se creará un widget Flash incluyendo sólo el reproductor/grabador de vídeo de la versión de Flex, comunicándolo mediante JavaScript con el resto de la web. Otro caso de preocupación, es el estado de la implementación del standard CSS3[3] en los navegadores. No todos los navegadores soportan CSS3, o no todos soportan aún todas sus funcionalidades. Por ello, se creará una hoja de estilos CSS2 (soportada por todos los navegadores) que se cargará cuando se detecte que un usuario utiliza un navegador que no soporta las funcionalidades usadas por Babelium en CSS3. Es probable que se pierdan algunos detalles vistosos o elegantes de la web que CSS2 no soporta, pero al menos el usuario podrá seguir navegando sin ningún problema.

Cuadro 4.20: Riesgo: Soporte HTML5

4.6. Resumen: herramientas utilizadas en el control del proyecto

- Pizarra: Objeto práctico, que ayuda a despejar la cabeza y planificar las cosas más claramente.
- Scrum: Metodología de desarrollo ágil de software.
- Evernote: Herramienta de gestión de tareas GTD.
- Catch: Herramienta simple y rápida para la toma de notas desde un dispositivo Android.
- Google Groups: Método de comunicación del grupo de Babelium Project.
- Mercurial: Control de versiones del proyecto.
- Google Code: Hosting de Mercurial que ofrece interfaz para planificación de proyectos ágiles.
- GMail: Gestor y organizador de emails.
- LaTeX: Herramienta de composición de textos con la que se ha redactado la memoria, usando la plantilla proporcionada por el grupo de software libre "itsas" de EHU.
- Google Docs: Creación de imágenes, esquemas y documentos.
- Dropbox: Herramienta online para albergar y compartir contenidos mediante la "nube".
- Gimp: Herramienta de retoque fotográfico.
- DIA: Utilizada para crear los diagramas de diseño.
- Wordpress: Blog para informar sobre HTML5.
- Openproj: Programa utilizado para generar los diagramas de gantt.

Capítulo 5

Especificaciones de Babelium Project

Contents

A lo largo de este capítulo se pretende mostrar las principales similitudes y diferencias entre la versión de Flex de Babelium Project y la versión en HTML5, con el fin de mostrar las características y resultados del proyecto que pretende migrarse. De esta forma, desarrolladores externos pueden hacerse una idea del tipo de migración que documenta esta memoria.

Mientras que en este capítulo únicamente se *mostrará* las tecnologías y arquitecturas de ambos sistemas, en capítulos posteriores se hará más hincapié y se profundizará en la elección de dichas tecnologías y en el proceso de migración paso a paso.

5.1. Babelium Project - Flex

A continuación se mostrarán dos subsecciones, una con las tecnologías empleadas por la versión a migrar, y otra explicando la arquitectura de sus sistema.

5.1.1. Tecnologías de lado cliente

Adobe Flex

Adobe Flex (hasta 2005 Macromedia Flex) es un término que agrupa una serie de tecnologías publicadas por Macromedia para dar soporte al despliegue y desarrollo de las *RIA*, basadas en su plataforma Flash. Flex creó su propio lenguaje basado en etiquetas XML para la creación de interfaces gráficas: MXML. Combinando código MXML, con scripts *Action Script*, Flex se convirtió en una herramienta a tener en cuenta en el desarrollo de portales web. Como Flex incluye todas las funcionalidades de Flash, también es posible recibir datos de un servidor de streaming con esta tecnología. Por todo lo anteriormente mencionado, se convirtió en la tecnología idónea para el desarrollo de Babelium.

Cairngorm mini-Framework

Toda aplicación distribuida[1], necesita tener una arquitectura bien definida para tener un código claro y de calidad que gestione tanto el cliente como la interacción con servicios externos. Para ello se utilizó **Cairngorm**, framework que ofrece la posibilidad de

[1] Aplicación con distintos componentes que se ejecutan en diferentes entornos, normalmente en distintos ordenadores conectados por red

implementar un patrón arquitectural MVC sobre nuestra aplicación de manera relativamente sencilla. Mediante el uso de este patron, nos permite crear aplicaciones flexibles y escalables. Además, gracias al patrón Delegate (del que Cairngorm también hace uso), nuestra aplicación puede acceder a servicios remotos de una manera estructurada y bien definida.

Utilizando Cairngorm en el cliente, pueden distinguirse distintos tipos de elementos que tomarán parte en la arquitectura de la aplicación:

- **Vistas:** componentes gráficos que contienen la GUI de la aplicación. Botones, paneles, formularios...
- **Eventos:** Cuando el usuario realiza distintas acciones, se disparan eventos con información sobre ellas. Ej: la pulsación de un botón.
- **Comandos:** En su interior contienen las acciones a realizar cuando se captura una acción disparada por el usuario.
- **Controlador:** Elemento único en el sistema. Asocia los eventos disparados con los comandos correspondientes, y los ejecuta en el momento indicado.
- **Modelo:** También único en el sistema. Aquí se almacenan todos los datos del usuario y sistema, para poder ser referenciados desde cualquier parte de la aplicación.
- **Delegados:** Encargados de hacer de pasarela entre los comandos y los servicios remotos.

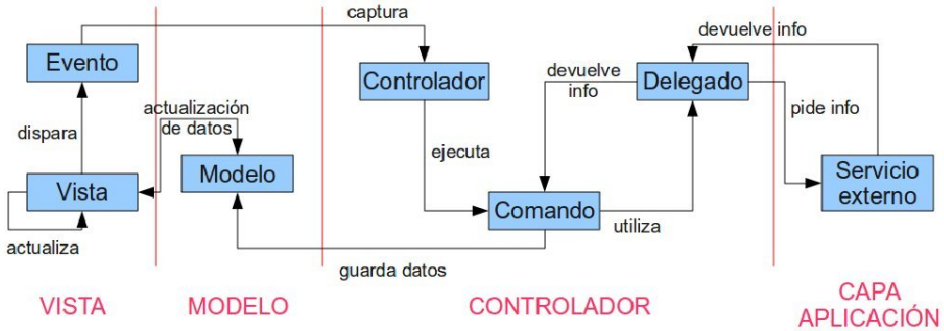


Figura 5.1: Resumen de esquema de interacción Cairngorm

Así, las clases de nuestra Vista quedan ligeras, pues el procesamiento de datos se realiza en las clases Comando. El Controlador de la arquitectura captura nuestros eventos

(o peticiones) y los enlaza con sus correspondientes comandos (o tratadores de petición). Los comandos a su vez realizan diversas acciones. En caso de necesitar información externa, se contacta con los delegados, los cuales piden la información requerida a los servicios externos y se la devuelven al comando para su procesamiento. En la figura 5.1 puede observarse un esquema de esta interacción.

Flash Player

Conocido y extendido plugin que hoy en día ya viene integrado en gran parte de los navegadores. Permite reproducir archivos .swf (*shockwave flash*) en el mismo navegador ofreciendo así infinitas posibilidades de enriquecimiento a la web.

5.1.2. Tecnologías de lado servidor

Red 5

Red5 es un servidor de código libre que ofrece la funcionalidad de streaming de vídeo. La aplicación está escrita en Java, con un servidor de aplicaciones Tomcat interno. La otra alternativa era Flash Media Server, pero es software privativo cuyo precio ronda los 1000$, así que fue inmediatamente descartado.

Red5 incluye varias aplicaciones de ejemplo fácilmente instalables mediante un instalador interno. Estas aplicaciones permiten observar ejemplos del funcionamiento de Red5, e incluso, con algunas de ellas se obtuvo más de lo necesario para poder realizar streaming de vídeo. Una razón más a favor de Red5.

En caso de necesitar nuevas aplicaciones en Red5, se pueden realizar usando tecnología Java junto con Spring Framework.

Utiliza el protocolo RTMP para intercambiar datos con Flash.

PHP

PHP es uno de los lenguaje de scripting más extendido en el entorno Web. Por su simplicidad, su alcance y su potencia, fue elegido como candidato a formar la capa del servidor de Babelium Project. Más tarde, encontrar librerías de conexión AMF (protocolo de comunicación con el cliente en Flex) escritas en PHP hicieron decantarse por esta opción, pues es un lenguaje que se aprende realmente rápido y ofrecía todo lo necesario para terminar la capa de servicios.

Zend Framework

Amplia y potente librería como extensión al lenguaje PHP. Zend nos ofrece un framework bastante completo para utilizar distintos servicios desde el lenguage PHP. Se utilizan módulos concretos del framework, como son Zend Mail (para envío de mails) o Zend AMF (para comunicar la aplicación en Flex con los servicios externos en PHP).

MySQL

Gestor de base de datos utilizado por Babelium. De los más usados por los desarrolladores de aplicaciones Web, gratuito, fácil de descargar, instalar y configurar... cumple todos los requisitos para integrarlo en nuestra aplicación. Mediante consultas SQL, este gestor almacenará y gestionará toda la información de la aplicación: usuarios, vídeos, subtítulos, preferencias...

Apache Web Server

Servidor web por excelencia. Atenderá peticiones HTTP para ofrecer a los usuarios el acceso a BP. Además, Apache alojará los servicios PHP y el framework Zend que recibirá y procesará las peticiones AMF encapsuladas sobre HTTP.

5.1.3. Resumen de tecnologías

En la tabla 5.1 se muestra un resumen de éstas.

Tecnología	Función
Lado Cliente	
Adobe Flex	Lenguaje de programación para el código de la parte del cliente.
Cairngorm	Mini-Framework que define una arquitectura MVC para la parte cliente; escalable y flexible.
Lado Servidor	
Red5	Servidor de archivos de vídeo que enviará datos en streaming y recibirá las grabaciones de los usuarios.
PHP	Lenguaje de scripting con el que se harán peticiones a la base de datos.
Zend Framework	Librería PHP con varias utilidades.
MySQL	Gestor de base de datos donde se guardarán todos los datos de Babelium.
Apache Web Server	Servidor web que alojará la aplicación y sus servicios.

Cuadro 5.1: Tabla de tecnologías de Babelium en su versión Flex

5.1.4. Arquitectura del sistema

La arquitectura del sistema la componen todas las tecnologías, servicios y servidores anteriormente mencionados. En la figura 5.2 se puede observar una imagen de la arquitectura general del sistema.

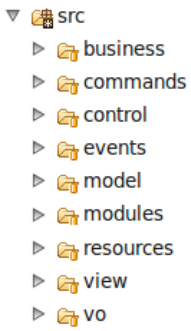
Como se puede observar, interactúan diversas tecnologías para ofrecer Babelium Project al cliente. Todo empieza cuando un usuario entra a su navegador web e introduce la URL de Babelium. En este momento, el navegador envía una petición HTTP al servidor web Apache de Babelium. El cliente recibe en respuesta un contenedor HTML junto con un archivo SWF que contiene el cliente de la aplicación web.

A partir de este momento el usuario puede navegar por la aplicación en local, mediante eventos y comandos de la arquitectura Cairngorm en Flex que se explicará más adelante. Cuando el usuario requiere información adicional, o un servicio externo, entra en juego AMF. Flex recoge la petición del usuario (o automática del sistema) en formato AMF, y se la envía al servidor Apache encapsulada en una petición HTTP. Apache a su vez se la envía a la librería ZendAMF que hace de *gateway*[2] entre Flex y PHP. Entonces Zend procesa la información y llama al servicio PHP correspondiente, que en caso de ser necesario, hará sus consultas SQL a la base de datos y devolverá el resultado.

Cuando el usuario quiera reproducir vídeo, sin embargo, entrará en juego el proto-

[2]Pasarela, punto de enlace.

colo RTMP a través del cual se comunicará con el servidor Red5 que le servirá o le permitirá grabar un vídeo.



Para una mejor comprensión de capítulos siguientes, entraremos un poco en detalle de la organización del código de la aplicación por directorios. Como se mencionó en el apartado anterior, el cliente sigue la arquitectura de Cairngorm, y por ello sigue una jerarquía de directorios que implementa el patrón propuesto por Cairngorm. A continuación se puede observar una imagen de los directorios seguido de una explicación de lo que alberga cada uno de ellos:

business: Contiene las clases "delegadas", que son las que se encargan de comunicarse con los servicios externos, y devolver los resultados a los comandos que piden información externa.

commands: Todos los comandos, tanto los externos como pedir información de ejercicios disponibles, como los internos como puede ser cambiar de un módulo a otro, se encuentran en esta carpeta. Un comando es una acción a realizar que necesite interacción entre módulos, modelos o servicios y no pueda tratarse desde la clase desde la cual se lanza.

events: Eventos personalizados y definidos por los programadores para que ejecuten diferentes comandos.

control: Aquí se encuentran las clases únicas[3] que controlan el "estado" de Babelium. Una de las más importantes es el *Controller*, que se encarga de hacer funcionar la arquitectura de Cairngorm. Recoge todos los eventos disparados por los módulos, y en función del tipo de evento dispara un comando u otro. Otras clases importantes que se encuentran aquí son el BabeliumBrowserManager, que se encarga de gestionar el *deeplinking*, y el CuePointManager que gestiona los cuepoints de los vídeos de cada módulo.

model: Contiene toda la información necesaria para que la aplicación funcione. Es un modelo de información a la que todos los módulos pueden acceder en cualquier momento.

modules: La única excepción a la organización de directorios propuesta por la arquitectura de cairngorm. Se creó para tener ordenados y organizados todos los componentes gráficos y módulos que componen Babelium.

[3]Son clases únicas (o singleton) aquellas de las cuales existe una única instancia en toda la aplicación.

resources: Imágenes, archivos de configuración, templates, skin...

view: La funcionalidad inicial de la carpeta *view* fue sustituida por *modules* (nombre más intuitivo) y view actualmente contiene clases para mejorar la apariencia visual de algunos componentes, como por ejemplo un TimeFormatter.

vo: La cual significa ValueObject y contiene los objetos y estructuras necesarias para interpretar los datos enviados/recibidos a los servicios.

src: Por último, la carpeta src global, contiene el Main que arranca la aplicación, así como archivos .xml de configuración de servicios.

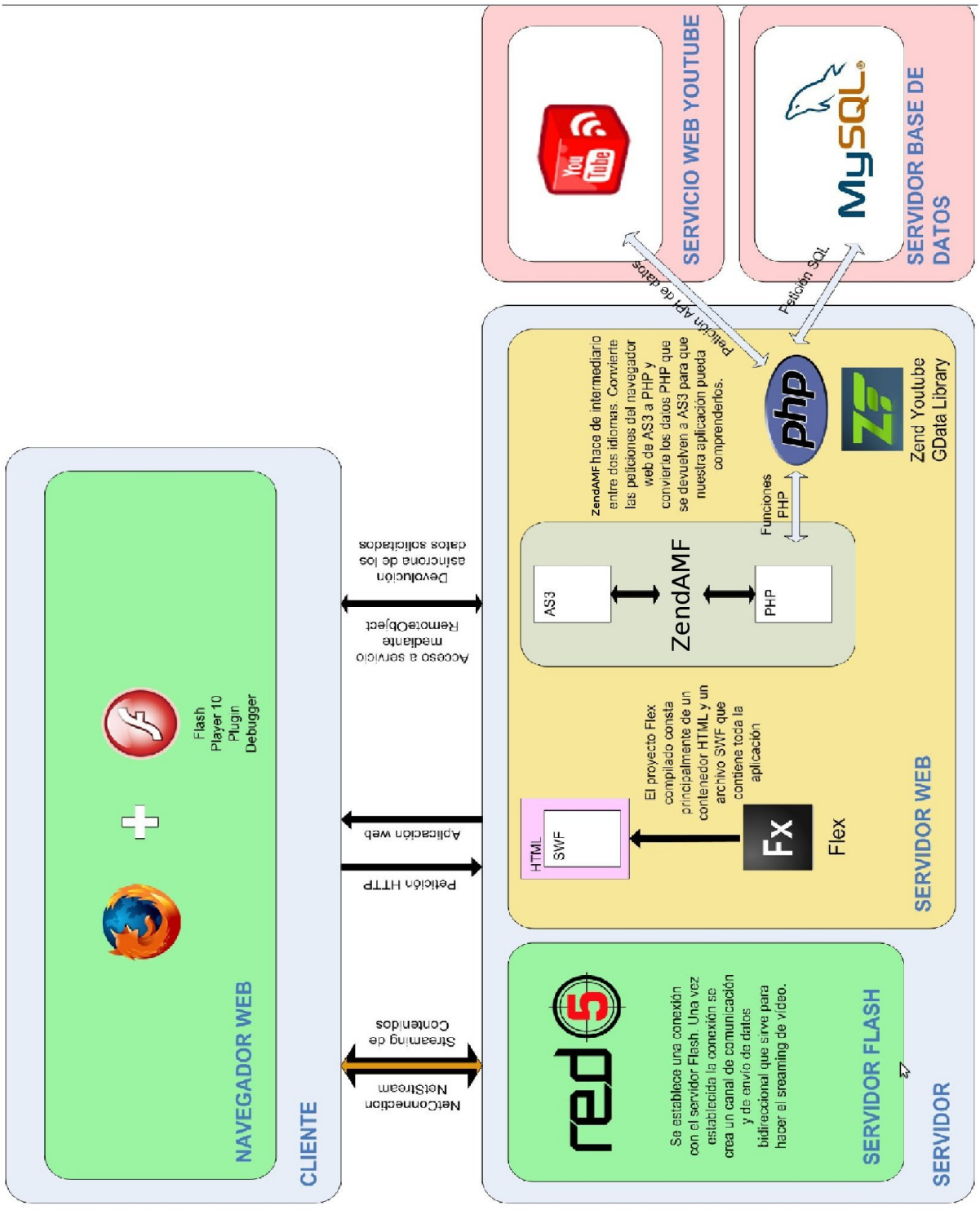


Figura 5.2: Arquitectura general de la aplicación

5.2. Transformación a HTML5

Como se ha visto en el apartado anterior, la aplicación en Flex puede dividirse en dos partes, cliente y servidor.

Cliente: Las tecnologías de la parte cliente son Flex (que integra MXML y ActionScript) y Cairngorm MVC (patrón modelo-vista-controlador también escrito en ActionScript).

Servidor: Las tecnologías del lado servidor la forman PHP y Zend Framework (obviando los servidores: Apache, MySQL y Red5).

Comunicación: La comunicación entre cliente y servidor se realiza mediante consultas AMF (encapsuladas dentro del protocolo HTTP).

A lo largo de esta sección se resumirán brevemente las transformaciones que sufren los 3 puntos antes mencionados en la versión de HTML5. Y como se mencionó al comienzo de este capítulo, se encontrará explicación y más información de la toma de decisiones en los posteriores.

5.2.1. Transformación de la parte cliente

Antes de proseguir, si aún hubiera dudas respecto al funcionamiento de Cairngorm aún después de haber leído la sección 5.1.1, está disponible en el anexo **??** información detallada y con un ejemplo real del funcionamiento avanzado de Cairngorm MVC dentro de Flex; recomiendo encarecidamente hacer una pausa y saltar a dicho punto en caso de que fuera necesario.

En resumidas cuentas, Cairngorm ofrece una abstracción a la capa vista para facilitar su gestión, como se muestra en la figura 5.3 [10]:

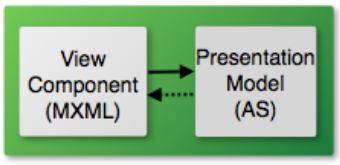


Figura 5.3: Abstracción que ofrece Cairngorm

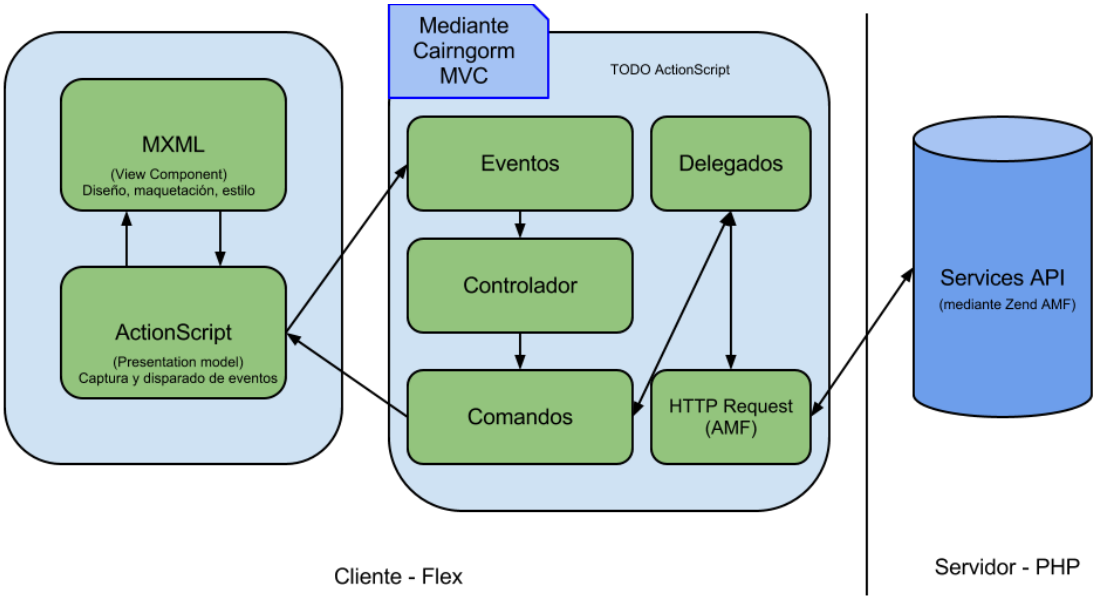


Figura 5.4: Resumen de la arquitectura interna del cliente en Flex

Adentrándonos un poco más en la arquitectura interna del cliente en Flex, en la figura 5.4 pueden verse los componentes principales que lo forman:

Código MXML: son, principalmente componentes para la maquetación y diseño de la vista. Así mismo pueden incluir en su interior scripts avanzados de ActionScript para la gestión de eventos.

Código ActionScript integrado en los componentes MXML: ActionScript es el lenguaje de scripting dentro de Flex. Dentro de los componentes MXML, realiza captura de eventos que estos lanzan y realiza cualquier acción pertinente.

Cairngorm MVC: framework escrito en ActionScript que implementa el patrón MVC y ofrece abstracción y escalabilidad al cliente.

Código ActionScript que implementa el patrón MVC de Cairngorm: eventos, comandos, controladores y delegados que se encargan de hacer la página web dinámica, atendiendo a las acciones del usuario y obteniendo información de servicios externos (servicios PHP en servidores remotos e información de la base de datos).

Gestión de contenidos: la gestión de contenidos en el cliente se realiza mediante el

diseño MXML integrado en el mismo y se actualiza mediante *DataBindings*[4].

Código MXML

El código MXML integra componentes de visualización: elementos de layout[5], componentes visuales básicos como formularios, capas, imágenes, etc.. hasta componentes más complejos como *DataTables* (tablas de datos: ordenables, modificables..), reproductores de vídeo (mediante streaming en el caso de Babelium), *ViewStacks*[6].

1. Layout: HTML5 introduce la denominada web semántica[7]. Mediante las nuevas etiquetas de maquetación (`header`, `footer`, `section`, `article`..) y las nuevas hojas de estilo CSS3[8] [11] se puede migrar fácilmente el layout de Flex. Más información sobre HTML5 y componentes semánticos en la sección 7.2.2.
2. Componentes básicos: HTML5 ya contiene soporte para todos los componentes básicos de Flex y sus estilos (Lista de similitudes entre estilos de Flex y HTML5 en el anexo A.2).
3. Componentes complejos: Para los componentes más complicados, existe una librería para JavaScript llamada JQuery[12] y JQuery UI que ofrece una amplia variedad de componentes visuales y efectos realmente útiles. Además, está ampliamente soportada por una gran comunidad que ha diseñado una gran cantidad de componentes adicionales fácilmente utilizables.

Código ActionScript integrado en los componentes MXML

El código ActionScript integrado en los componentes MXML se utiliza para la captura y disparado de eventos; por ejemplo cuando un usuario pincha sobre un componente, o cuando selecciona un elemento. Para controlar dichos eventos, se utiliza, de manera similar que ActionScript integrado en MXML, JavaScript[9] integrado dentro de las etiquetas HTML.

Ejemplo de botón en MXML+ActionScript que comienza una grabación al pulsarlo (evento click que dispara una acción ActionScript):

[4]Actualización dinámica de la vista implementando el patrón Observer: cuando un dato cambia la vista se actualiza

[5]Maquetación; alineación y colocación de componentes

[6]Panel con contenido variable en función de la sección en la que se encuentre

[7]Otorga a las etiquetas de HTML un significado semántico.

[8]Hojas de estilo para estilizar y definir mejor el layout y diseño de los componentes HTML

[9]Lenguaje de scripting que se ejecuta en el cliente (navegador del usuario)

```
<mx:Button id="startRecordingButton" label="Start recording"
  click="onStartRecordingClicked(event)"/>
```

Y su equivalente en HTML5+JavaScript:

```
<button id="startRecordingButton" value="Start recording"
  onclick="onStartRecordingClicked(event)"/>
```

Cairngorm MVC

En Flex, Cairngorm MVC era el framework en ActionScript que se encargaba de facilitar un patrón de diseño para la captura de eventos y la respuesta a éstos. Cómo uno de los principales objetivos de este proyecto, era mantener la arquitectura de la aplicación similar, se dedicó una parte de este proyecto a realizar un *port*[10] de la librería Cairngorm MVC al lenguaje JavaScript, para así poder seguir usándola en el cliente HTML5.

De esta forma, mientras en Flex encontrábamos: MXML + ActionScript integrado + Cairngorm MVC (ActionScript); en el proyecto en HTML5 mantenemos la misma arquitectura de aplicación, pero en lenguajes diferentes: HTML5 + JavaScript integrado + Cairngorm MVC (JavaScript).

Se darán más detalles del motivo y el modo de migración de esta librería en posteriores capítulos.

Código ActionScript que implementa el patrón MVC de Cairngorm

Como se ha mencionado anteriormente, aquí se encuentra el núcleo de la aplicación: eventos que disparan acciones del usuario en la página web, controladores que asocian comandos a cada tipo de evento, comandos que reaccionan y responden al evento en cuestión, delegados para buscar información en servicios externos.. en resumidas cuentas, implementa en ActionScript el patrón MVC utilizando la librería Cairngorm.

Una vez más, gracias a la migración de Cairngorm a JavaScript, todo el núcleo de la aplicación pudo ser migrado a JavaScript, siguiendo el mismo patrón MVC de Cairngorm.

[10]Migración de una lenguaje a otro

Gestión de contenidos

Aquí reside una de las grandes diferencias entre la versión de Flex y HTML5. Mientras que en Flex, el contenido se encuentra en el cliente (la aplicación Flash SWF) y mediante Cairngorm sólo se recoge información para actualizar los contenidos; en HTML5 todos los contenidos se encuentran en la parte del servidor (los motivos se contemplarán en posteriores capítulos) y se utiliza Cairngorm para obtener dichos contenidos y actualizar la Web.

Dicho de otra forma, se creó un *Content Manager* en JavaScript, que mediante el patrón MVC de Cairngorm, es capaz de obtener nuevos contenidos para la web del servidor (en respuesta a eventos del usuario) y actualizar la web principal, de forma dinámica y sin recargos de página.

5.2.2. Transformación de la parte servidor

En la versión anterior de Babelium, en el servidor sólo se encontraban los servicios de la aplicación; servicios escritos en PHP que acceden a la base de datos para devolver información a la aplicación cliente. En la versión en HTML5, dado que esta es la parte servidor y es independiente del cliente, se mantuvieron todos los servicios en las mismas condiciones. Misma base de datos y mismo lenguaje (PHP).

Sin embargo, se hicieron modificaciones a los servicios para mejorar la accesibilidad de la web, de los servicios y de la información asociada a los mismos, así como para la gestión de sesiones, que antes se realizaban en el cliente Flex y ahora en el servidor en PHP:

1. Instalación de los servicios a modo de API REST[11], para que la información de la que dispone Babelium pudiera ser reutilizable y hacer la aplicación más escalable.
2. Implementación de medidas de seguridad y gestión de sesiones de usuario en los servicios.

En esta parte arriba mencionada, recibí mucha ayuda por parte de Inko Perurena, quien implementó gran parte de las tareas mencionadas con resultados muy positivos. Dado que la API de Babelium Project era una parte de Babelium realmente importante (tanto para la versión de Flex como HTML5), Inko realizó el trabajo antes de que yo comenzará el proyecto, permitiéndome reutilizarla después en la versión HTML5.

[11]Una API es un conjunto de servicios disponible para que desarrolladores externos puedan crear fácilmente extensiones o subprogramas que utilicen la misma información que Babelium Project.

Además, antes toda la carga dinámica de contenidos se realizaba en el cliente. Es decir, antes la aplicación era un SWF Flash que el usuario se descargaba y navegaba sobre ella, y se utilizaba Cairngorm **sólo** para recoger información del servidor. Ahora Cairngorm lo que recoge son los contenidos, y en el servidor se encuentra una infraestructura escrita en PHP que sirve los contenidos, es decir, las páginas web y sus secciones que están precompiladas como plantillas HTML5 utilizando Smarty.

Por ejemplo, antes en Flex, cuando un usuario accedía al módulo practice, para mostrar la lista de ejercicios disponibles al usuario, el cliente en Flex obtenía de los servicios la información de los ejercicios, es decir, un objeto por cada ejercicio indicando título, duración, vídeo, etc.. y el cliente en Flex, se encargaba de maquetar esa información y mostrarla gráficamente en la pantalla.

Por el contrario, ahora en HTML5, cuando el usuario accede al módulo practice, para mostrar la lista de ejercicios disponibles, el cliente realiza una petición al servidor, y en el mismo servidor, se obtiene la información y se maqueta mediante HTML+CSS devolviendo al cliente el resultado ya estructurado y gráficamente visible, que sólo tiene que actualizar en la página.

Gestor de plantillas: Smarty

Como se acaba de explicar, la gestión de contenidos en Flex se realizaba en el cliente, en un archivo SWF compilado que contenía **toda** la estructura de la página. Sin embargo, el protocolo HTTP para HTML funciona de forma distinta, y por defecto funciona de manera pregunta y respuesta, el cliente pide una página y el servidor se la devuelve.

Es por ello que se tuvo que tomar una decisión para gestionar los contenidos. Decidimos que la mejor manera de hacerlo era mediante un gestor de plantillas. Las plantillas son fragmentos de HTML+CSS que se guardan en ficheros y un gestor se encarga de cargarlas y mostrarlas en pantalla. La primera decisión que hubo que tomar, fue si implementar el gestor de plantillas en el Cliente (JQuery tiene un gestor de plantillas implementado en JavaScript) o en el servidor (encontramos la librería Smarty para PHP especialmente atractiva).

Decidimos implementar el gestor de contenidos en el servidor, principalmente porque implementar un gestor de plantillas en el cliente, supondría utilizar JavaScript y por lo tanto, la CPU y memoria del cliente para procesar la web, mientras que si gestionamos las plantillas en el servidor, no se consumen recursos del computador del cliente. Dado que aún hay usuarios utilizando ordenadores y navegadores antiguos, no se vio como solución viable implementar la gestión de plantillas en el cliente, quedando Smarty como la solución idónea.

Más información sobre las plantillas Smarty en la sección 8.1.4 y sobre sus funcionalidades avanzadas en el apéndice A.2.

5.2.3. Transformación en la comunicación cliente-servidor

Como parte similar, al igual que antes se encargaba Cairngorm (ActionScript) de realizar la comunicación del cliente con el servidor, ahora se encarga Cairngorm (JavaScript). Sin embargo, antes la hacía encapsulando el mensaje en una trama AMF dentro del protocolo HTTP; mientras que ahora se realiza directamente por HTTP con un sistema de seguridad basado en Tokens. Se entrará en detalle y se explicará mejor en el capítulo 8.3.

5.2.4. Limitaciones tecnológicas

Muy a nuestro pesar, con el estado actual de implementación de HTML5, ha sido imposible migrar (por ahora) el 100 % de la aplicación. Los navegadores actuales aún no tienen implementada la captura de audio y vídeo (micrófono y webcam) por lo que los usuarios no podrían grabar ejercicios, y resulta ser la funcionaliad principal de Babelium. Es por ello que se decidió separar el reproductor/grabador de vídeo del resto del código en Flex, y en la versión HTML5 se sigue utilizando como widget en Flex, que se comunica con el resto de la web mediante una API de acceso para javascript.

5.2.5. Resumen de tecnologías

En la tabla 5.2 se muestra un resumen de las tecnologías utilizadas en la versión HTML5.

Tecnología	Función
Lado Cliente	
Adobe Flex	Únicamente utilizado en el reproductor/grabador de vídeo y audio.
Cairngorm JavaScript	Mini-Framework que define una arquitectura MVC para la parte cliente; escalable y flexible.
HTML5+CSS3	Diseño, estilo y maquetación de la página web y sus componentes
JavaScript	Contiene los eventos y comandos que implementan la arquitectura Cairngorm para obtener respuesta a eventos del usuario
JQuery	Encargada de implementar los componentes complejos (como los *DataTables*) en HTML y sirve de utilidad para las clases de JavaScript que implementan Cairngorm, así como para añadir efectos y hacer la navegación por la web más agradable.
Lado Servidor	
Red5	Servidor de archivos de vídeo que enviará datos en streaming y recibirá las grabaciones de los usuarios.
PHP	Lenguaje de scripting con el que están escritos los servicios y la infraestructura del servidor.
Smarty	Gestor de *templates* (plantillas) para PHP.
Zend Framework	Librería PHP con varias utilidades.
PHP REST API	API que implementa el patrón de conexión REST.
MySQL	Gestor de base de datos donde se guardarán todos los datos de Babelium.
Apache Web Server	Servidor web que alojará la aplicación y sus servicios.

Cuadro 5.2: Tabla de tecnologías de Babelium en su versión HTML5

5.3. Resumen de transformaciones

En la figura 5.5 pueden observarse las transformaciones resultantes.

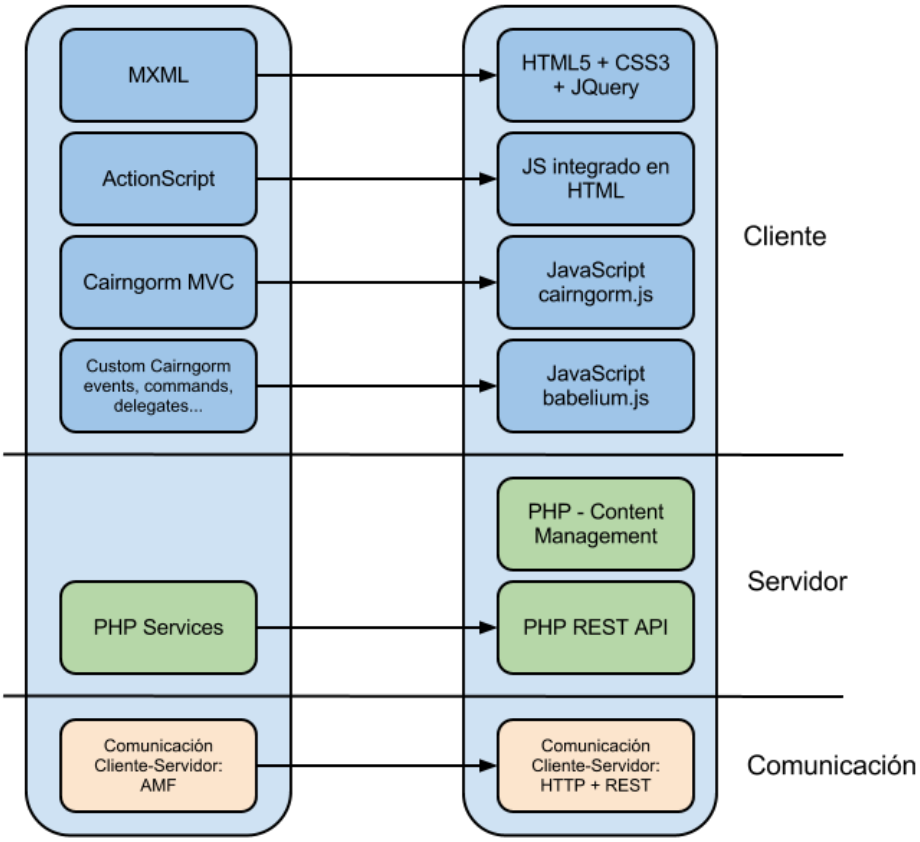


Figura 5.5: Resumen de transformaciones Flex a HTML5

Gracias al *Google Closure Compiler*[12] se han podido *compilar* todos los scripts JavaScript que forman cairngorm en la librería **cairngorm.js** y todos los scripts JavaScript que implementan el patrón Cairngorm para Babelium en la librería **babelium.js**.

A continuación se muestran a grandes rasgos las similitudes y discrepancias entre ambas versiones.

[12]Compilador y empaquetador de javascript para hacer el código más eficiente y compacto

5.3.1. Resumen de similitudes

- La apariencia de la página es prácticamente la misma
- El reproductor/grabador de vídeo es el mismo (sólo que exportado independientemente)
- La arquitectura de la aplicación en el cliente es exactamente la misma (distintos lenguajes, pero misma arquitectura)
- Se sigue utilizando el patrón MVC en el cliente
- Se reutilizan los servicios escritos en PHP anteriormente
- Se sigue utilizando MySQL como gestor DB y Red5 como servidor de streaming

5.3.2. Resumen de discrepancias

- El cliente está escrito en HTML5 + CSS3 + JavaScript + JQuery (en lugar de ActionScript + MXML)
- Cairngorm se ha reescrito a JavaScript (del su versión oficial en ActionScript)
- Se accede a los servicios mediante una API REST en lugar de mediante peticiones AMF
- Se controlan las sesiones de los usuarios en el servidor mediante las *Sessions* de PHP
- Se ha creado una infraestructura en el servidor que se encarga de recoger la información y servir al usuario la sección (o trozos de la sección) en la que se encuentra
- Antes la aplicación era un SWF que se descargaba y se navegaba sobre él, ahora la página web es un script PHP con una infraestructura en su mismo lenguaje que en su primera carga, descarga las plantillas de la sección en la que se encuentra y la aplicación cliente compilada en JavaScript. A partir de ese momento, la navegación por la web se realiza de manera dinámica y se cambia el contenido de la página mediante AJAX[13], realizando peticiones de plantillas para los contenidos nuevos.

[13] *Asynchronous JavaScript And XML*, se utiliza para obtener información de una servicio externo desde un script en JavaScript

Capítulo 6

Estrategias y antecedentes de migración

Contents

A lo largo de este capítulo se mostrarán las distintas estrategias y/o herramientas disponibles para hacer una migración de software, y la decisión tomada junto a su motivo.

Actualmente existen gran cantidad de aplicaciones que ofrecen opciones de migración de Flash a HTML5. Estas herramientas se centran en la automatización de un proceso: teniendo el código fuente de una aplicación Flash o directamente su archivo binario (un archivo SWF), intentan generar un equivalente utilizando sólo HTML5. A continuación se muestra un resumen de dichas aplicaciones y sus objetivos, estudiados y documentados en el artículo escrito por GHyM [7]:

6.1. Traducir ActionScript a JavaScript

Adobe Wallaby. "Wallaby"[13] es el nombre en clave de una tecnología experimental que convierte el arte y animación contenida en Adobe®Flash®Proffesional (FLA) en HTML5. Aunque hay gran cantidad de funcionalidades sin implementar, es un buen punto de partida para diseñadores que quieren una versión beta de su obra en HTML5.

Jangaroo. Jangaroo compiler [14] traduce un subconjunto de ActionScript 3 a JavaScript 1.5, soportado por la mayoría de navegadores. Es una aplicación OpenSource que añade estructuras de JavaScript para simular un subconjunto de ActionScript 3 que no está nativamente implementado, como: *classes*, *packages*, *private members* y *static typing*.

FalconJS. FalconJS es un compilador experimental de MXML y ActionScript a HTML y JavaScript, construido sobre Falcon [15], el futuro compilador de aplicaciones Flex. Falcon parsea el código ActionScript y crea un *Abstract Syntax Tree* (AST), que es reducido a *bytecode* que puede ser interpretado por una ActionScript Virtual Machine (AVM2) como es Flash Player. FalconJS reemplaza el *bytecode generator* por un *backend* que genera JavaScript. A día de hoy es solamente un prototipo interno, pero parece ser la salida por la que apuesta Adobe.

6.2. Interpretar los archivos binarios Flash (swf) en JavaScript

Este enfoque consiste en maquinas virtuales escritas en JavaScript capaces de interpretar el mismo bytecode que interpreta el plugin de Flash. Existen distintas alternativas

como es Google Swiffy [16], pero lamentablemente a día de hoy soportan solamente una muy vieja y limitada versión de la máquina virtual de Flash (la versión 8).

6.3. Ingeniería Inversa

Utilizando la herramienta de ingeniería inversa Sothink SWF to HTML5 [17] es posible generar el equivalente HTML5 de un archivo SWF. Sin embargo, a día de hoy solamente es utilizable para convertir animaciones básicas de Flash a HTML5, que no requieren de ninguna interacción de JavaScript.

6.4. Migración Dirigida por Modelos

Migración Dirigida por Modelos (*Model Driven Migration*), es la automatización mediante modelado de software y técnicas MDA (*Model Driven Architecture*) de la migración del código origen al código destino. Consiste en extraer el modelo de la aplicación para luego realizar distintas transformaciones sobre él hasta obtener automáticamente gran porcentaje del código de la aplicación en el lenguaje destino [18]. La figura 6.1 muestra el proceso:

Como se puede observar, primero se obtiene, partiendo del código, el modelo de la aplicación en un lenguaje específico de dicha plataforma. Acto seguido, mediante ingeniería inversa, se realiza una transformación de ese modelo a un *metamodelo*[1]. Después, mediante transformaciones de modelo a modelo (M2M) y modelo a texto (M2T) de MDA se obtiene como resultado una transición, del metamodelo, a un modelo de las tecnologías destino, y por último, al código resultante en dichas tecnologías.

Las **ventajas** de la migración guiada por modelos frente a la migración manual son varias. Entre ellas podemos destacar:

- **Reutilización del proceso:** una vez el proceso está definido y todas transformaciones programadas, cualquier cambio en el modelo de la aplicación puede reflejarse inmediatamente en el código volviendo a ejecutar el proceso de transformaciones ya definido. De la misma forma, el proceso es reutilizable en proyectos futuros de las mismas características.
- **Mayor abstracción a la hora de diseñar:** de la misma forma que los lenguajes de programación y los compiladores, al código ensamblador; los modelos, lenguajes

[1]Representación de un modelo de código al nivel conceptual más alto

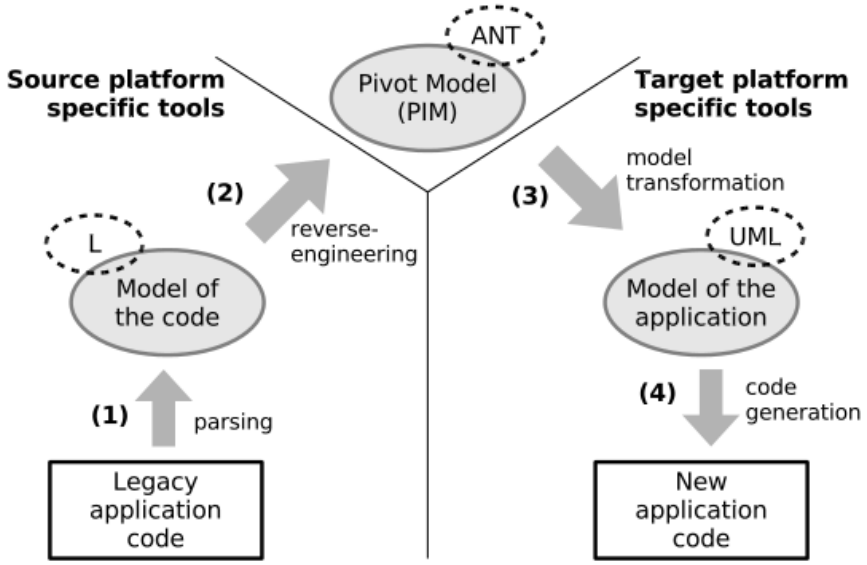


Figura 6.1: Proceso de una migración dirigida por modelos (MDM)

de modelado y lenguajes de transformaciones de modelo pretenden ofrecer una mayor abstracción a la hora de diseñar software. Este proceso ofrece generación automática de gran parte del código.

Y **desventajas**:

- **Gran coste de aprendizaje:** hay que invertir una gran cantidad de horas en aprendizaje de este nuevo paradigma.
- **Gran coste de modelado y diseño de procesos para las transformaciones.**

La figura 6.2 ilustra la diferencia temporal de la migración utilizando las dos técnicas distintas en un proyecto de gran magnitud.

Mientras que migrando manualmente, se consigue código destino desde el comienzo, la migración dirigida por modelos conlleva un coste inicial de puesta apunto. Sin embargo, también puede apreciarse que tras el inicial *setup*, la generación de código es mucho más rápida, y además, el mismo proceso puede utilizarse para modificaciones en ese mismo proyecto, o reutilizarlo en proyectos futuros de las mismas características.

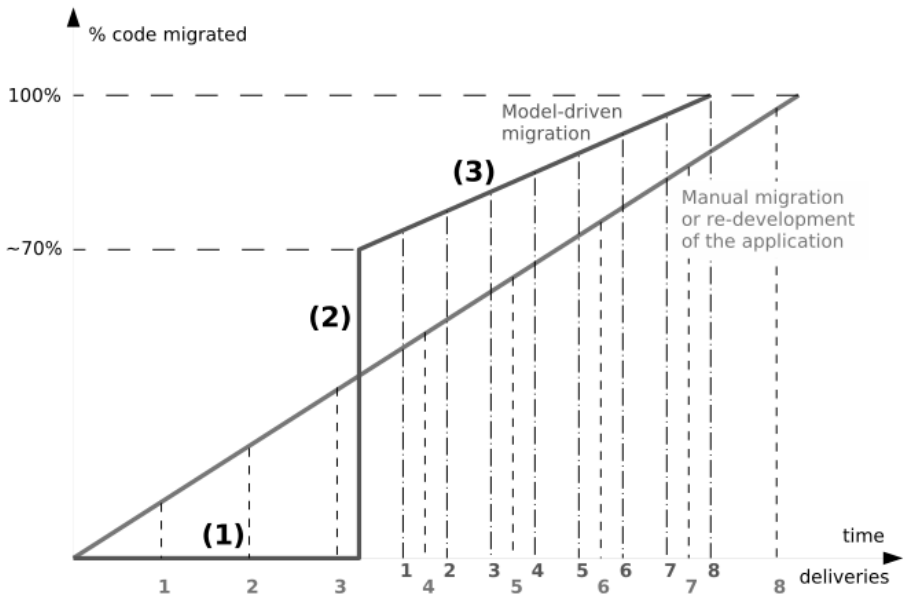


Figura 6.2: Representación temporal de dos tipos de migración

6.5. Conclusiones

Cómo se ha visto en las subsecciones anteriores, ninguna de las herramientas de migración automatizada está en un estado funcional y usable. Las que traducen código de Flex a JavaScript sólo traducen subconjuntos muy reducidos, y teniendo en cuenta que la aplicación está estructurada utilizando un framework complejo como es Cairngorm, no es viable utilizar un traductor.

Lo mismo pasa con los interpretadores de SWF en JavaScript o la ingeniería inversa para construir una aplicación HTML5. Aunque son alternativas interesantes, ofrecen resultados muy pobres.

Por otra parte, la Migración Dirigida por Modelos representa una alternativa interesante, sin embargo, se evaluó que conllevaría un coste excesivo diseñar modelos y transformaciones de lenguajes tan complejos como MXML+ActionScript y Cairngorm, como para realizarlo dentro de un PFC de un año de duración.

Además, es extremadamente complicado realizar transformaciones y modelos para un conjunto de tecnologías HTML5 cuando nunca antes se ha trabajado sobre ellas. Es por ello que decidimos finalmente, encaminarnos por otro tipo de migración, y realizar así una aplicación en HTML5 para tener como base, y en un futuro si que podrían realizarse

modelos y transformaciones para futuros proyectos, pero eso no estará cubierto por este PFC.

Como ninguna solución anterior era factible, y ninguna tenía en cuenta que quizá la aplicación en Flex utilizaba un framework arquitectural, me decanté por un modelo de migración que mantuviera la misma arquitectura en la aplicación cliente. Uno de los pasos para ello, fue traducir, como ya se ha mencionado, el Framework de Cairngorm de Flex a JavaScript, de modo que facilitara enormemente el proceso manual de migración y ofreciera un entorno de programación muy similar al original en Flex.

Aclaradas las opciones de migración disponibles, se procedió a realizar una migración manual, dividida en dos tareas principales:

- Identificar y definir los distintos componentes de Flex y sus posibles equivalencias en HTML5
- Crear y seguir un plan de migración de Flex a HTML5 una vez identificado la forma en la que se migrarían los distintos componentes.

Esas dos tareas se explicarán en detalle en los próximos dos capítulos, el primero para explicar como representar o transformar Flex a HTML5, y el segundo explicando el plan de iteraciones definido y la forma en la que se migró Babelium Project.

Capítulo 7

Prácticas y patrones de migración

Contents

Una vez definida la estrategia a emplear a lo largo de la migración: migrar manualmente la aplicación manteniendo la misma arquitectura en ambos lenguajes; esta sección pretende resumir algunos puntos clave y pautas que se han seguido a lo largo de la migración. Dicho de otra forma, esta sección resumirá las decisiones que se han tomado y la forma en la que se han migrado los distintos componentes que contenía Flex, pudiendo servir así como modelo para migraciones de otro proyectos Flex.

En la sección 5.2 se resumen por encima las transformaciones realizadas al código origen para migrarlo a HTML5, en esta sección se pretende profundizar en dichas transformaciones aclarando las decisiones tomadas y especificando la forma en la que se implementaron.

Se dedicó una larga e inicial iteración de investigación, con la finalidad de indagar en las posibilidades de HTML5, y las opciones/equivalencias que había a la hora de migrar el código. Como resultado de dicha investigación, se desarrollo *Cairngorm JS*, se definieron ciertas prácticas y se decidió el uso de ciertas tecnologías para representar la aplicación en el lenguaje destino. Esa investigación es la que quedará representada en las siguientes subsecciones, que se centrarán en:

Videoplayer. Modificaciones realizadas al videoplayer para su reutilización en HTML5.

MXML. Estrategias seguidas para representar el layout y diseño de la aplicación en Flex en HTML5+CSS3.

View Components. Implementación de componentes complejos (*ViewStacks*, *Paginations*, *Ratings*...) en HTML5.

Cairngorm JS. Migración del mencionado framework Cairngorm MVC de ActionScript a JavaScript para mantener exactamente la misma arquitectura de aplicación.

ActionScript. Estrategias seguidas para la implementación del código ActionScript que responde a las acciones del usuario.

7.1. Videoplayer

El reproductor de vídeo es el componente principal de Babelium Project. Gracias a él se puede realizar streaming y captura de vídeo del usuario. Dado que estas funcionalidades, aún no están totalmente soportadas en HTML5, se decidió utilizar el reproductor de vídeo de Flex como componente independiente y añadirle una interfaz de comunicación con JavaScript.

7.1.1. Independizar el componente

Esta fase se realizó realmente rápido, ya que el reproductor de vídeo lo realicé como PFC de la Ingeniería Técnica y ya estaba diseñado de forma totalmente independiente al resto de la aplicación Flex, por lo que crear un nuevo proyecto en Flex y compilar únicamente el reproductor de vídeo no fue una tarea demasiado complicada.

Se creó un *branch* (rama) nuevo en el repositorio donde se guardo el proyecto "embeddable videoplayer" que contenía el código y recursos necesarios para compilar el reproductor. Cuya compilación dió como resultado el reproductor de vídeo como widget Flash que se puede observar en la figura 7.1.

Una vez independizado, integrarlo en una web HTML es muy sencillo, tan sólo hay que añadir las siguientes líneas a la página:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="babeliumPlayer" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/[..]/swflash.cab">
  <param name="movie" value="util/swf/babeliumPlayer.swf" />
  <param name="quality" value="high" />
  <param name="bgcolor" value="#ffffff" />
  <param name="flashVars" value="" />
  <param name="wmode" value="window" />
  <param name="allowScriptAccess" value="sameDomain" />
  <embed src="util/swf/babeliumPlayer.swf" quality="high"
    bgcolor="#ffffff" flashVars=""
    width="100%" height="100%" name="babeliumPlayer"
    align="middle" wmode="window"
    play="true" loop="false" quality="high"
    allowScriptAccess="sameDomain"
    type="application/x-shockwave-flash"
    pluginspage="http://www.adobe.com/go/getflashplayer">
  </embed>
</object>
```

Figura 7.1: Videoplayer como widget Flash en una página HTML5.

7.1.2. Interfaz de comunicación

Pese a que independizar el reproductor de vídeo, fue una tarea rápida y sencilla, aún faltaba por aclarar el modo en el que se iba a comunicar la web con el reproductor. Antes, dado que toda la parte del cliente estaba escrita en Flex, y el reproductor estaba integrado en ella, era muy sencillo realizar una llamada al reproductor, tan sólo había que hacer referencia al reproductor y usar sus métodos.

```
1 <videoPlayer:VideoPlayerBabelia id="VP" [..]>
```

```
1 VP.play();
```

Sin embargo, ahora el cliente de la web estaría escrito en JavaScript, mientras que el reproductor sería un widget en Flash; por lo que sería necesario un interfaz para controlar el reproductor desde JavaScript.

Aquí entra en juego el *External Interface* de Flex. Nos permite declarar métodos externos que se llamarán desde JavaScript y realizarán acciones en Flex. Gracias al *ExternalInterface* todos los métodos y propiedades públicas del reproductor pudieron ser fácilmente exportadas para ser usadas desde JavaScript. El código Flex necesario para que esto sea posible es algo tan sencillo como:

```
1 ExternalInterface.addCallback("play", VP.play);
```

Además, se añadió una función JavaScript que es invocada en cuanto el reproductor termina de cargarse, pasando el mismo reproductor como parámetro de la función:

```
1 ExternalInterface.call("onConnectionReady", ExternalInterface.objectID);
```

De esta forma desde JavaScript se puede controlar el reproductor de manera muy sencilla:

```
1  function onConnectionReady(playerId)
2  {
3    var videoPlayer = null;
4
5    if (navigator.appName.indexOf("Microsoft") != -1){
6      videoPlayer = window[playerId];
7    } else {
8      videoPlayer = document[playerId];
9    }
10
11   if (!videoPlayer)
12     return;
13
14   videoPlayer.play();
15 }
```

A continuación se muestra una lista con los atributos y operaciones públicas del reproductor y su función, invocables desde JavaScript para ofrecer un control total:

Propiedades	Descripción
(write-only) arrows : Boolean	Muestra/oculta el panel de flechas y del rol que está hablando en cada momento.

autoPlay : Boolean	Si está a verdadero, comienza la reproducción de un vídeo según se cargue (sin tener que darle al botón play)
autoScale : Boolean	Define si el vídeo a reproducir se va a auto-escalar hasta cubrir al 100 % todo el campo de visión del reproductor (si está en verdadero), o si se va a escalar manualmente siguiendo su *aspect-ratio* hasta que se ajuste lo mejor posible (si está en falso).
(write-only) controlsEnabled : Boolean	Habilita/deshabilita los controles de reproducción (play, pause, stop y seek).
(read-only) duration : Number	Devuelve la duración total del vídeo que se está reproduciendo en ese momento.
secondSource : String	Asigna un segundo vídeo para reproducir en paralelo en el modo PLAY_BOT_STATE.
seek : Boolean	Habilita/deshabilita la posibilidad de hacer seek en un vídeo.
(write-only) skin : String	Cambia la skin del reproductor de vídeo al fichero cuyo nombre se guarda en esta propiedad.
state : int	Esta propiedad contiene el estado actual en el que se encuentra la máquina de estados del reproductor. Modificarla cambiaría automáticamente el estado del mismo.
streamSource : String	Servidor + protocólo desde donde se realizará el streaming de vídeo.
(read-only) streamTime : Number	Segundo actual en el que se encuentra la reproducción del vídeo actual.
(write-only) subtitles : Boolean	Activa/desactiva los subtítulos en el vídeo.
subtitlingControls : Boolean	Muestra/oculta los controles de subtitulación usados en el módulo subtitles.
videoSource : String	Nombre de el vídeo que se está reproduciendo. Modificarlo cambiaría el vídeo en reproducción automáticamente por su nuevo valor.
highlight : Number	Resalta el reproductor cuando habla el rol escogido para grabar un ejercicio.

Cuadro 7.1: API de propiedades del reproductor

Seguido de una API de operaciones para el control externo del reproductor:

Operaciones	Descripción
disableControls() : void	Deshabilita los controles de reproducción (play, pause, stop, seek).
enableControls() : void	Habilita los controles de reproducción (play, pause, stop, seek).
endVideo() : void	Termina la reproducción del vídeo. Tras esta operación se ha de volver a cargar un vídeo si se desea volver a reproducir.
muteRecording(flag:Boolean) : void	Silencia la captura de sonido del micrófono si se está grabando, o el segundo vídeo en paralelo si se están reproduciendo dos al mismo tiempo.
muteVideo(flag:Boolean) : void	Silencia el vídeo principal que se está reproduciendo en ese instante.
pauseVideo() : void	Pausa la reproducción del vídeo.
playVideo() : void	Carga y comienza la reproducción del vídeo que contiene la propiedad videoSource.
removeArrows() : void	Elimina todas las flechas de roles que existan en el panel de flechas.
resumeVideo() : void	Reanuda la reproducción del vídeo desde donde se dejó.
seekTo(time:Numer) : void	Mueve en el tiempo la reproducción del vídeo hasta el segundo *time*.
setArrows(list:ArrayCollextion, selectedRole:String) : void	Asigna al panel de flechas la lista (*list*) de flechas que tiene que añadir, así como el rol que se a es c digo (*selectedRole*) para poner sus flechas en rojo y las otras en negro. *arrows:* ArrayCollection[time:Number,role:String]
setSubtitle(text:String) : void	Muestra el texto *text* en la caja de subtítulos del reproductor.
startTalking(role:String, duration:Number) : void	Muestra durante *duration* segundos el nombre *role* hablando en el RoleTalkingPanel.
stopVideo() : void	Detiene la reproducción del vídeo y lo devuelve al segundo 0. stopVideo() = pauseVideo() + seekTo(0);
toggleControls() : void	Cambia de activado a desactivado (y viceversa) los controles de reproducción de vídeo (play, pause, stop, seek).

Cuadro 7.2: API de operaciones del reproductor

7.2. Estructura y estilo de la página

Una parte que aunque simple, es realmente importante, es el *layout* o la estructura (gráfica) de la página; es decir, donde y cómo se colocan los elementos.

7.2.1. Layout en Flex

En Flex el layout se construye a mediante *containers* (contenedores). Por ejemplo en el caso que nos ocupa, Babelium Project, el layout estaba construido con los siguientes elementos:

BorderContainer. Es un contenedor genérico. Se le puede especificar si el contenido va a estar alineado horizontal o verticalmente, y además, se le pueden especificar cualquier tipo de estilo similar al CSS para cambiar cualquier propiedad (color, ancho, largo..) de los bordes y el contenedor.

VGroup. Es un contenedor básico, sin estilos, donde todos sus elementos se alinean verticalmente, ocupando el 100 % de su ancho.

HGroup. Al igual que el anterior, es un contenedor básico, pero en este caso sus componentes se alinean horizontalmente, ocupando el 100 % de su altura.

Group. El componente que engloba los dos anteriores, se le puede especificar manualmente la orientación de sus componentes; un contenedor genérico sin estilos.

A continuación puede verse un ejemplo de layout en flex que muestra 4 botones alineados horizontalmente [19]:

```
1 <s:HGroup>
2   <s:Button label="Button 1"/>
3   <s:Button label="Button 2"/>
4   <s:Button label="Button 3"/>
5   <s:Button label="Button 4"/>
6 </s:HGroup>
```

Figura 7.2: Layout horizontal básico en Flex

Y en la figura 7.2 se muestra su resultado:

Y para hacerse una idea de como se podría diseñar una web con un estilo elegante, a continuación se muestra un trozo de código que contiene un layout horizontal con un poco de estilo en el fondo y los bordes [20]:

```
<s:BorderContainer
    backgroundColor="red" cornerRadius="10"
    borderStyle="inset" borderWeight="4" >
  <s:layout>
    <s:HorizontalLayout
 paddingLeft="5" paddingRight="5"
 paddingTop="5" paddingBottom="5"/>
  </s:layout>
  <s:Button label="Button 1"/>
  <s:Button label="Button 2"/>
  <s:Button label="Button 3"/>
  <s:Button label="Button 4"/>
</s:BorderContainer>
```

Cuyo resultado se puede observar en la figura 7.3.

Figura 7.3: Layout horizontal básico en Flex

Resumiendo, para hacer el *layout* o estructura de la página, en Flex se utilizan contenedores, que requieren 2 tipos de propiedades:

- ➔ **Estilo:** Tamaño, diseño (color de fondo, color del borde, tipo de borde..) y posición del contenedor.
- ➔ **Orientación:** Orientación en la que se alinean los componentes internos que contenga; Vertical u Horizontal.

En HTML5 el estilo se soluciona mediante CSS3 (hojas de estilo) y la orientación de los componentes dentro de un contenedor, se aplica utilizando las nuevas *flexible boxes*[21] que implementa HTML5.

Pero antes de adentrarnos en cómo estilizar u orientar un contenedor en HTML5, veamos los tipos de contenedores disponibles.

7.2.2. Contenedores en HTML5

En HTML5 se implementan los nuevos *contenedores semánticos*. Inicialmente, en HTML existía el **div**, también llamado *capa* o *layer*. Una *capa* es un contenedor genérico, similar al *BorderContainer* de Flex. Permite agrupar todo lo que hay en su interior en un contenedor, y además, mediante CSS3 se le permite dar estilo. Sin embargo, con la *web semántica*[1] se incluyeron nuevos contenedores, cuya funcionalidad es exactamente la misma que la de una *capa*, pero pretende añadir semántica a la web para poder ordenar los contenidos de la web en bloques o contenedores que tengan *sentido*.

Los contenedores que otorgan semántica a la web, en resumidas cuentas, son los siguientes:

header. un bloque de contenido que representa la cabecera de la web si se encuentra en el más alto nivel, o de otro contenedor en caso contrario.

footer. viene a ser lo mismo que el anterior, pero representa el pie de página, o pie de contenedor.

nav un bloque para una sección de enlaces, el menú de enlaces navegables por la web, por ejemplo; cualquier sección que sirva para navegar por las secciones de la web, o por web externas.

aside. representa una sección de la web que tiene poco que ver con el resto o que contiene contenido secundario. Un ejemplo puede ser la publicidad, el *motd* (messages of the day) etc..

section. sección genérica de la página, o subsección dentro de un artículo o una sección más general.

article. sección más concreta de la página. Puede representar bien un artículo, un comentario, o hasta un *widget*.

[1] Contenedores con connotaciones semánticas para una mejor estructura de la web.

Utilizando y anidando cualquier numero de los contenedores anteriores, y posteriormente, añadiéndoles su correspondiente estilo y diseño, se puede estructurar una web completa en HTML5 que sea semánticamente correcta.

Para entender mejor lo mencionado anteriormente, vamos a tomar como ejemplo la web de Babelium Project, observable en la figura7.4:

Figura 7.4: Apariencia de la web de Babelium Project en Flex

Se puede observar que la web tiene una cabecera en la que aparece un menú navegable con las secciones principales, otro menú navegable con opciones como login, logout y selección de idioma, un logo y un buscador. Después está la sección en la que nos encontramos actualmente, la cual tiene una cabecera indicando el nombre de la sección, y subsecciones que contienen (en este caso) vídeos para reproducir. La figura 7.5 representa estos componentes por bloques.

Y teniendo en cuenta los contenedores mencionados anteriormente, podría traducirse al layout que refleja la figura 7.6.

Figura 7.5: Layout de la web de Babelium Project en Flex

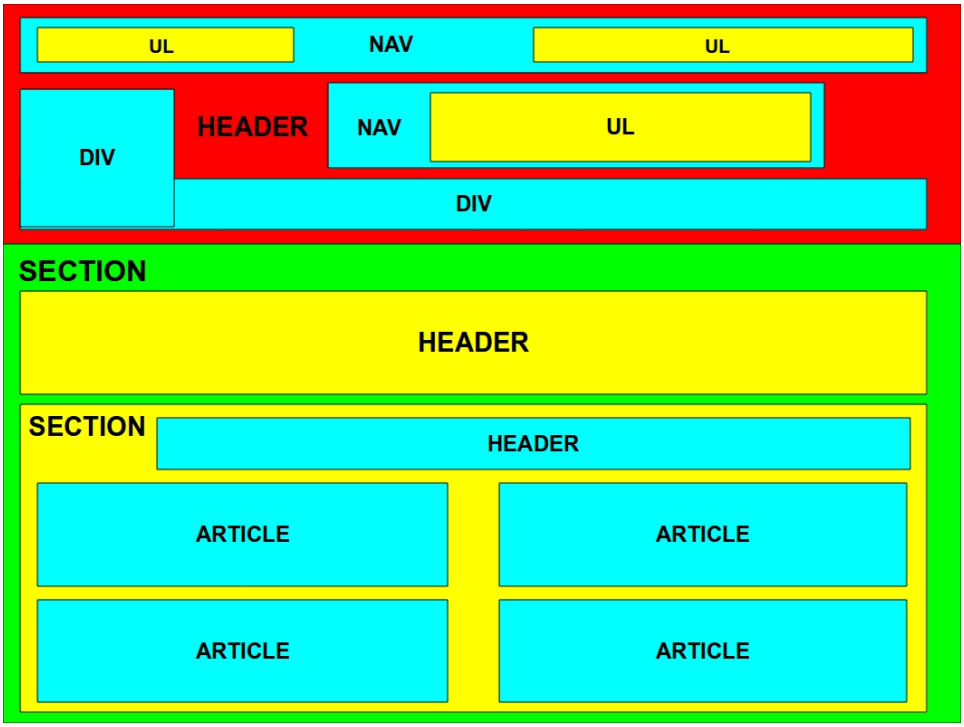


Figura 7.6: Esquema del layout semántico de la web de Babelium Project

7.2.3. Estilo de los componentes en HTML5

En Flex se utilizaba un lenguaje muy similar al CSS3 para estilizar los componentes, es decir: para cambiar su tamaño, color, bordes, .. etc. En HTML5 esto se consigue mediante las hojas de estilo CSS3. Veámoslo mejor con un ejemplo, para ello tomemos como referencia un extracto del último layout estilizado en Flex mostrado en el ejemplo del apartado anterior:

```
<s:BorderContainer
    backgroundColor="red" cornerRadius="10"
    borderStyle="inset" borderWeight="4" >
  <s:layout>
    <s:HorizontalLayout
 paddingLeft="5" paddingRight="5"
 paddingTop="5" paddingBottom="5"/>
  </s:layout>
  [..]
</s:BorderContainer>
```

En HTML5 el CSS3 se puede integrar dentro de los componentes, al igual que en Flex, o puede crearse en un fichero/script separado y adjuntarse a los componentes en base a sus identificadores. Por ejemplo, para observar que la sintaxis de los estilos no difiere demasiado de los utilizados en Flex, veamos como quedaría el ejemplo traducido a HTML5 con CSS3 integrado (tomando **div** como contenedor):

```
<div style="background-color: red; border-radius: 10px;
     border-style: inset; border-width: 4px;
     padding-left: 5px; padding-right: 5px;
     padding-top: 5px; padding-bottom: 5px;">
  [..]
</div>
```

Al igual que en Flex, si le añadimos un identificador al contenedor, se puede crear un script (por ejemplo: *style.css*) que contenga los estilos de los componentes identificados por ID:

```
<div id="contenedorPrueba">
  [..]
</div>
```

Y su estilo asociado en lenguaje CSS3:

```
#contenedorPrueba
{
  background-color: red;
  border-radius: 10px;
  border-style: inset;
  border-width: 4px;
  padding-left: 5px;
  padding-right: 5px;
  padding-top: 5px;
  padding-bottom: 5px;
}
```

En el apéndice A.2 se profundizará más sobre como hacer y estilizar un layout en HTML5+CSS3 y las similitudes entre el estilizado de componentes en Flex y en CSS3.

7.2.4. Orientación y alineación de los componentes de un contenedor

Una gran ventaja que tenían los contenedores era el ordenado y alineación automática de componentes dentro de un contenedor mediante los H/VGroup o los Horizontal/Vertical Layouts. En HTML4 (o inferior), no existía esta comodidad, los componentes había que alinearlos de forma distinta, pero para mantener la mayor fidelidad posible con la estructura de Flex, se hizo uso de los nuevos *Flexible Box Model*.

Éstos cumplen exactamente la misma función que los HGroup o VGroup (e incluso estilizados como un BorderContainer), tan sólo hace falta añadir propiedades CSS a uno de los contenedores mencionados en la sección 7.2.2 y automáticamente se convertirá en un contenedor similar a cualquiera de los mencionados en Flex.

Para ello, a continuación se definen dos estilos: HGroup y VGroup; que asignados a cualquier contenedor, lo convierten automáticamente en su equivalente del mismo nombre en Flex.

```
.HGroup, .VGroup
{
  display: box;
  box-align: stretch;
}

.HGroup { box-orient: horizontal; }

.VGroup { box-orient: vertical; }
```

La propiedad *stretch* hace que los componentes en su interior se estiren hasta ocupar el 100 % de su ancho o algo (en función de la orientación). Mencionado esto, para convertir cualquier contenedor en un HGroup, por ejemplo, tan sólo hay que asignarle el estilo:

```
<div class="HGroup">
  [..]
</div>
```

Para un mejor ejemplo, vamos a ver un layout jerárquico de Flex que contiene un contenedor con alineación vertical y en su interior dos contenedores horizontales, conteniendo cada uno 2 botones:

```
<s:VGroup>
  <s:HGroup>
    <s:Button label="Button 1"/>
    <s:Button label="Button 2"/>
  </s:HGroup>
  <s:HGroup>
    <s:Button label="Button 3"/>
    <s:Button label="Button 4"/>
  </s:HGroup>
</s:VGroup>
```

Y a continuación, su equivalente en HTML5+CSS3, tomando **div** como un contenedor genérico, pudiendo ser cualquier otro mencionado en la sección 7.2.2:

```
<div class="VGroup">
  <div class="HGroup">
    <button value="Button 1"/>
    <button value="Button 2"/>
  </div>
  <div class="HGroup">
    <button value="Button 3"/>
    <button value="Button 4"/>
  </div>
</div>
```

Existen otras propiedades como **box-align** o **box-pack** que permiten alinear los componentes internos de un contenedor al inicio, centrado o final (ya sea vertical u horizontalmente), pero estos aspectos avanzados se tratarán en el apéndice A.2 con más detalle.

7.3. View Components

La mayoría de los componentes básicos de Flex están ampliamente soportados en HTML5, como son: los formularios (botones, radio buttons, textfields, textareas..), enlaces, imágenes, etc.. Sin embargo, componentes algo más complejos no están soportados, por ejemplo:

ViewStacks: También conocidos como *TabPanels* o *TabBars*, son aquellos contenedores que permitencambiar su contenido dinámicamente. Aunque normalmente suelen tener pestañas para alterar su contenido, no son esenciales. En la siguiente imagen se puede observar un ejemplo de los *motd* (messages of the day):

Figura 7.7: Ejemplo de ViewStack o TabPanel

Paginations: Su función es similar a la de los *ViewStacks*, dividir todo el contenido en bloques e ir mostrando partes diferentes; salvo que la utilidad de este componente, es separar largas listas de resultados en listas más pequeñas y fácilmente navegables mediante una lista de páginas:

Figura 7.8: Ejemplo de pagination

DataTables: Son tablas complejas que permiten ordenación en tiempo real de los datos y búsquedas sobre ellos:

Ratings: Este componente, a diferencia de los anteriores, no es un contenedor pensado para mostrar dinámicamente contenido distinto. Su funcionalidad es bastante más simple. Es un componente que permite, mediante estrellas, mostrar la puntuación de algún contenido, y así mismo, ofrece la posibilidad de poder puntuar uno mismo y salvar la puntuación en la base de datos para luego mostrar la media:

Figura 7.9: Ejemplo de DataTables en Flex

Figura 7.10: Ejemplo de rating

Todos estos componentes no están incluidos en el *standard* HTML5, para poder implementarlos hay que utilizar alguna librería JavaScript. Hay scripts JavaScript que implementan por separado cada uno de esos componentes, pero para el proyecto nos decantamos por componentes implementados sobre un framework, ya que esto implica soporte y mantenimiento por parte de la comunidad. Los frameworks Javascript de creación de componentes UI (User Interface) más extendidos son::

JQuery UI. Extension del popular framework JQuery, que añade infraestructura para el fácil diseño de componentes UI; que contiene tanto widgets (tabs, buttons, progressbar, dialogs, sliders, datepicker, autocomplete..), como interacción con widgets y el entorno (draggable, droppable, resizable, selectable).. todo ello soportado en todos los navegadores.

YUI. Al igual que JQuery UI, Yahoo! UI (YUI) contiene prácticamente los mismos widgets y ofrece gran cantidad de facilidades para el desarrollo de widgets y web interactivas.

Cualquiera de los dos frameworks hubiera servido para el desarrollo de una web interactiva, pero dado que ya se había escogido JQuery como motor JavaScript de la web, nos decantamos por JQuery UI.

7.3.1. ViewStacks con JQuery UI

Los ViewStacks son fácilmente implementables mediante JQuery. Sólo es necesario un contenedor que contenga en su interior:

- Una lista de pestañas (opcional)
- Un contenedor por cada pestaña (opcional también, ya que los contenedores se pueden cargar mediante AJAX)

Por ejemplo, teniendo el siguiente contenedor que contiene una lista de pestañas y 3 contenedores adicionales en su interior:

```
<div id="pruebaTabs">
  <ul>
    <li><a href="#tabs-1">Nunc tincidunt</a></li>
    <li><a href="#tabs-2">Proin dolor</a></li>
    <li><a href="#tabs-3">Aenean lacinia</a></li>
  </ul>
  <div id="tabs-1">
    <p>Proin elit arcu, rutrum commodo, vehicula tempus, [..]</p>
  </div>
  <div id="tabs-2">
    <p>Morbi tincidunt, dui sit amet facilisis feugiat,  [..]</p>
  </div>
  <div id="tabs-3">
    <p>Mauris eleifend est et turpis. Duis id erat.  [..]</p>
    <p>Duis cursus. Maecenas ligula eros, blandit nec, [..]</p>
  </div>
</div>
```

Tan sólo hay que invocar JQuery (*$(**id**).tabs()*) en el inicio de la página para convertir dicho contenedor genérico en un ViewStack:

```
$(function() {
  $( "#pruebaTabs" ).tabs();
});
```

Cuyo resultado se refleja en la figura 7.11.

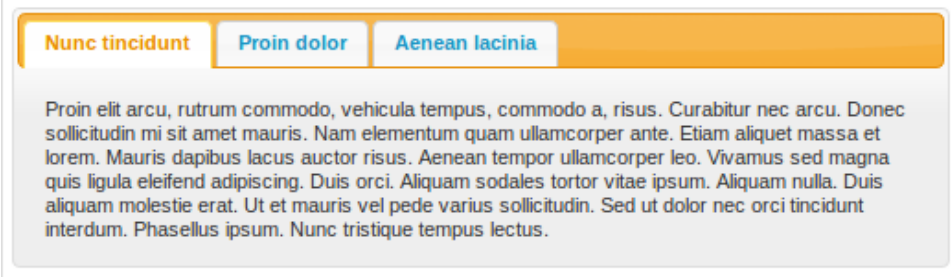


Figura 7.11: Ejemplo de ViewStack o TabPanel

7.3.2. Pagination con JQuery

Se utilizó, para el caso que nos ocupa, un plugin de JQuery llamado jPList. jPList es un plugin que de manera sencilla, ofrece **paginación**, **ordenación** y **filtrado** de cualquier conjunto de estructurase en una web, como por ejemplo: *divs* (contenedores), *ul* (listas), *tables*..

Un ejemplo del resultado que ofrece se puede observar en la figura 7.12.

En Babelium Project en HTMl5 se hizo uso de este plugin para mostrar una lista paginada, ordenada y filtrable de todos los ejercicios que se podían realizar y/o evaluar en la web, ejemplo observable en la figura 7.13

Para más información sobre este plugin se puede consultar su web y su documentación [22].

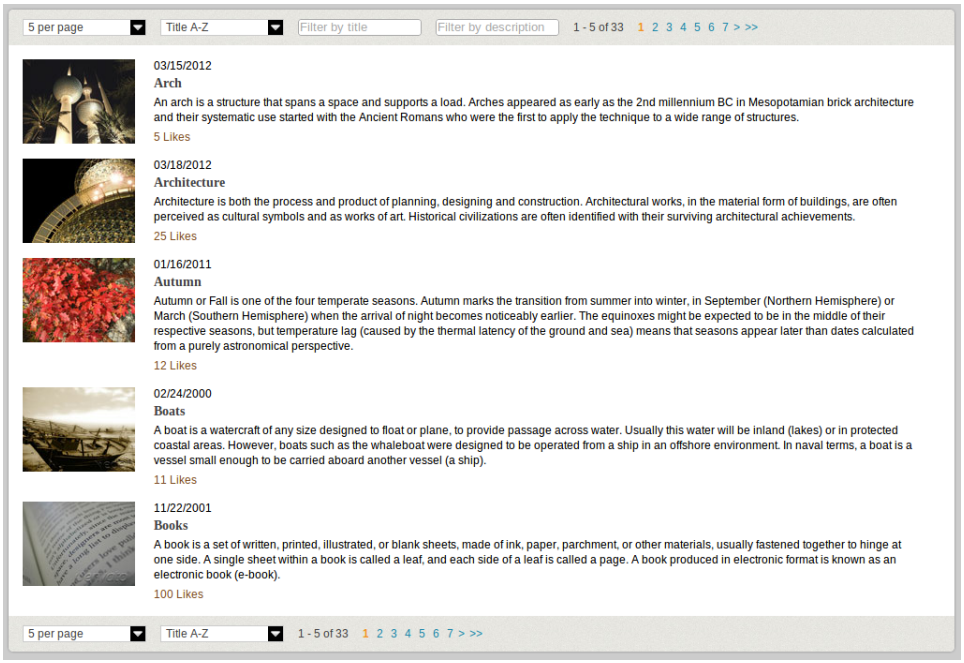


Figura 7.12: Ejemplo de paginación mediante JQuery y jPList

Figura 7.13: Ejemplo de paginación mediante JQuery y jPList

7.3.3. DataTables con JQuery

De forma similar que para el apartado anterior, en este caso se utilizó un plugin para JQuery llamado DataTables [23], que como su nombre bien indica, se encarga de hacer tablas de datos con las siguientes características adicionales: ordenación, filtrado y paginación; al igual que el anterior.

Puede que en este punto surjan dudas como: *¿porqué utilizar dos plugins con la misma funcionalidad?*, cuya respuesta será más obvia después de observar en la figura 7.14 la forma en la que este plugin visualiza los datos:

Figura 7.14: Ejemplo de DataTables mediante JQuery y DataTables

Mientras que el plugin jPList ha sido usado para paginar y ordenar contenido más estilizado, como una lista de vídeos, este plugin ha sido utilizado para visualizar datos en formato tabla, por ejemplo, el historial de actividades del usuario, que puede observarse en la figura 7.15.

Figura 7.15: Ejemplo de DataTables mediante JQuery y DataTables

7.3.4. Ratings con JQuery

Este es el widget más distinto de los demás, pues no hace de contenedor, sino que más bien es una utilidad en sí misma. Se implementa de una forma realmente sencilla sobre JQuery, mediante el plugin jRaty.

Es tan simple como coger un contenedor HTML:

```
1 <div id="rating"></div>
```

Y aplicarle en la carga de página una función de JQuery:

```
1 $('#rating').raty();
```

Lo que convierte el contenedor de ID rating en un widget de puntuación:

Figura 7.16: Widget de puntuación mediante JQuery y JRaty

7.4. Cairngorm JS

Dejando ahora un poco de lado los widgets y la estructura de la página, tras haber visto como integrar y controlar un widget Flex desde JavaScript, cómo hacer un layout y estilizarlo de forma similar a como se hace en Flex, y cómo crear widgets más complejos con la librería JQuery y sus plugins; esta sección se centra más en cómo se ha implementado el patrón controlador MVC en JavaScript, para poder responder de una manera fácil y ordenada a los eventos generados por el usuario: pulsación de un enlace o botón, envío de formulario, etc... cualquier tipo de navegación por la página.

Existen gran cantidad de frameworks para implementar el patrón MVC en JavaScript, a continuación se resume una breve lista de ellos con sus pros y contras[24]:

Backbone.js. Es uno de los frameworks más apoyado por la comunidad, bastante complejo y completo, sin embargo, deja bastante que desear a la hora de programar de manera abstracta, pone bastantes restricciones.

SproutCore 1.0. Framework utilizado por Apple en su iniciativa iCloud. Tiene soporte para *binding*, una comunidad solida y un montón de funcionalidades. Sin embargo, es un framework con una curva de aprendizaje muy lenta, complejo al principio, y obliga a NO utilizar HTML para el layout de la web.

Javascript MVC. A pesar de tener una gran comunidad, y una gran infraestructura, el nombre de este framework es demasiado genérico; pues es en realidad un framework escrito en Ruby que compilado genera una infraestructura en JavaScript.

GWT Google Web Toolkit es otro framework similar a SproutCore o JavaScript MVC. En este caso hay que programar sobre Java y compilado genera un proyecto Web en HTML+JS. Este tipo de frameworks, aunque interesantes, desvían demasiado el proyecto de rumbo.

Ember.js A plena vista parece un framework muy completo, sin embargo, es el equivalente al SproutCore 2.0 y aún es muy nuevo y con poca documentación y soporte de la comunidad.

Recordando que la finalidad de este proyecto es ofrecer en el mayor grado posible una transparencia en la migración, y que el grado de aprendizaje y el tiempo necesario para su implantación de cualquiera de los frameworks iba a ser muy alto, se decidió implementar Cairngorm MVC en JavaScript. A primera vista, puede parecer que escribir un framework propio en lugar de utilizar uno ya existente sea una locura, sin embargo, se llevo a cabo por las siguientes razones:

1. Todos los desarrolladores de Babelium ya están familiarizados con el funcionamiento del framework.

2. Es un framework bastante simple, que ofrece una infraestructura básica, por lo que implementarlo en JavaScript no sería muy pesado.

3. Adobe Open Source [25] tiene gran parte del código de Cairngorm MVC en Flex disponible online, por lo que migrarlo a JavaScript sería aún menos costoso.

En las próximas subsecciones se explicará el funcionamiento interno de Cairngorm, los elementos que lo componen y cómo se han migrado de ActionScript a JavaScript.

7.4.1. Estructura interna de Cairngorm

En capítulos anteriores, por ejemplo en la sección 5.1.1 se habla de la utilidad de Cairngorm. Cairngorm ofrece una infraestructura para poder crear de manera sencilla una arquitectura MVC en la aplicación. En la figura 7.17 puede observar la estructura interna de clases de Cairngorm y su funcionamiento:

Hallar la lista de clases, su funcionalidad y su implementación ha sido posible gracias al Open Source de Adobe.

Los siguientes elementos son los que componen Cairngorm prácticamente en su totalidad:

FrontController: es un objeto que tiene una tabla de asociaciones entre tipos de evento y comandos. Una vez se dispara un evento, si el evento es uno de los que el controlador soporta, este controlador es el encargado de ejecutar el comando asociado a dicho tipo de evento.

EventDispatcher: instancia única, tiene una tabla de asociaciones entre tipos de evento y controladores encargados de ejecutarlos. Cuando un evento se dispara (*.dispatch()*) la llamada siempre llega a esta instancia única, éste busca en la tabla el controlador asociado al evento, y envía la petición al controlador (FrontController).

Event: un objeto con únicamente dos atributos (tipo y *data*) y un método (dispatch). El atributo tipo es el identificador del evento mientras que el data contiene datos, por ejemplo un usuario. Cuando se llama al método dispatch, el EventDispatcher recoge el evento y se encargará de gestionarlo.

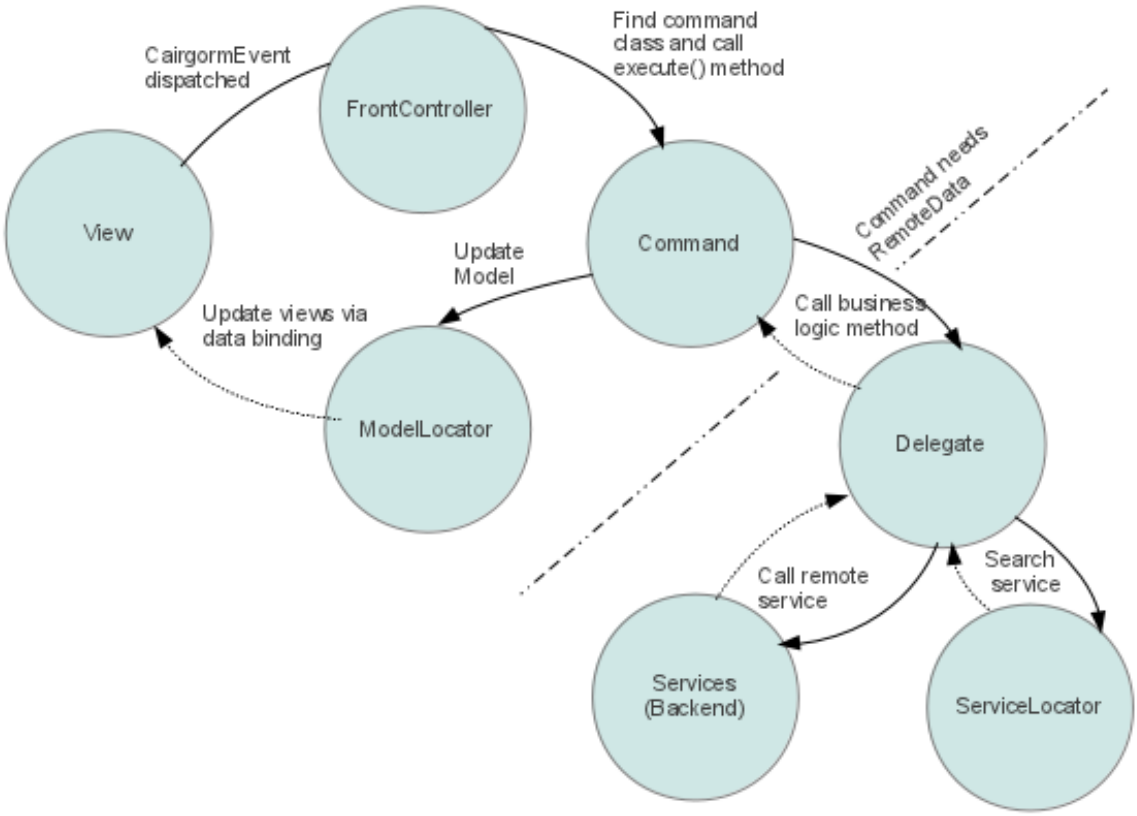


Figura 7.17: Estructura interna de Cairngorm

Command: un comando con un único atributo (*data*) y un método (execute). Cuando un controlador recibe un evento que puede manejar, crea un command nuevo conteniendo el *data* del Evento, y después se llama al método execute de dicho comando.

VO: son Value Object, o dicho de otra forma, objetos simples con atributos, sin métodos. Sirven para definir objetos con datos de usuario, de cuentas, de listas de ejercicios... y se utilizan para modelar los campos *data* de los Eventos y los Comandos.

ServiceLocator: instancia única. Utilizado como buscador de servicios, en nuestra implementación de Cairngorm JS, sólo disponible para servicios HTTP.

HTTPServices: una lista de servicios HTTP disponibles. Tienen una tabla de asociaciones idServicio -¿ Servicio HTTP y un método getService que devuelve el servicio solicitado (si existiera).

HTTPService: son objetos con un *gateway*[2] y ID de servicio como atributos. Este objeto tiene un método call que recibe como parámetro argumentos para la llamada, y un objeto de clase *responder*. El HTTPService realizara la llamada al *gateway* especificado con los argumentos especificados y devolverá la respuesta al método onResult (en caso de éxito) u onFault (en caso de error) del objeto *responder* pasado como parámetro.

7.4.2. Migrando Cairngorm a JavaScript

Teniendo la lista de clases que forman Cairngorm MVC y el código de su implementación básica en Flex, no debería ser muy difícil migrar el código a JavaScript. Sin embargo, el lenguaje JavaScript no está pensado (inicialmente) para funcionar con clases y herencia.

Los principales problemas que se encontraron a la hora de migrar el framework a JavaScript, fueron la creación de clases y la herencia de las mismas, y la conexión con los servicios externos.

Clases y herencia en JavaScript

Como se acaba de mencionar, JavaScript no está pensado para programar con clases y herencia, sin embargo, hay scripts y frameworks que ofrecen esa posibilidad.

En Babelium Project se hizo uso de un script implementado por John Resig [26], escritor de varios libros en JavaScript y colaborador de JQuery, entre otros. El script se llama Base.js y su utilización es muy simple y eficiente. El siguiente ejemplo es la creación de una clase en JavaScript utilizando Base.js:

```
1 var Person = Class.extend({
2   init: function(isDancing){
3     this.dancing = isDancing;
4   },
5   dance: function(){
6     return this.dancing;
7   }
8 });
```

Siendo *init* el constructor. *Class* viene a ser el objeto padre, como Object en Java

[2]Puerta de enlace mediante URL con un servicio externo

y la mayoría de los lenguajes. Y si por ejemplo, quisieramos crear una nueva clase que hereda de la nueva Person, tan sólo habría que extenderla de la misma forma:

```
var Ninja = Person.extend({
  init: function(){
    this._super( false );
  },
  dance: function(){
    // Call the inherited version of dance()
    return this._super();
  },
  swingSword: function(){
    return true;
  }
});
```

Pudiendo ser ahora ambas clases fácilmente utilizables como si de una clase en Java se tratase:

```
var p = new Person(true);
p.dance(); // => true

var n = new Ninja();
n.dance(); // => false
n.swingSword(); // => true
```

Gracias a esta clase, el framework Cairngorm se migró realmente rápido. A continuación se puede observar un ejemplo de la implementación de la clase Event de Cairngorm, ya que es la más simple y su código muy corto:

```
Cairngorm.Event = Class.extend(
{
  /**
   * Constructor
   * @param type = String
   */
  init : function ( type, data )
  {
    this.type = type;
    this.data = data != null ? data : {};
  },

  /**
   * Dispatch Cairngorm Event
   */
  dispatch : function ()
```

```
17   {
18     return Cairngorm.EventDispatcher.dispatchEvent(this);
19   }
20 });
```

Comunicación con los servicios externos desde JavaScript

Se resolvió utilizando AJAX mediante JQuery. Aprovechando que utilizabamos JQuery como motor para JavaScript, se incluyó su implementación de AJAX dentro de Cairngorm para implementar los HTTPServices. Hacer una llamada a un servicio externo resultó ser tan sencillo como especificar la URL de destino y los *callbacks* en caso de éxito o error:

```
1 $.ajax(
2 {
3   // Target url
4   url : src,
5   // timeout in millis
6   timeout : 5000,
7   // The success call back.
8   success : responder.onResult,
9   // The error handler.
10  error : responder.onFault
11 });
```

Al igual que con Cairngorm, JQuery se encargaría de realizar la llamada al servicio externo, recibir el resultado y llamar al comando (*responder*) ya fuera en caso de éxito o error.

Creación de Cairngorm JS

Por último, una vez migradas todas las clases que forman la estructura de Cairngorm a JavaScript, se hizo uso de Google Closure Compiler[3] [27] para unificar todas las distintas clases en un archivo **.js**. La función del Closure Compiler, es parsear el código JavaScript, analizarlo, eliminar el código muerto y reescribirlo minimizándolo y optimizándolo en la medida de lo posible.

Gracias al Closure Compiler se redujeron todas las clases que forman Cairngorm (14 ficheros que ocupaban un total de 22,9 KiB) a un fichero *cairngorm.js* de 1,9 KiB de tamaño.

[3]Optimizador de JavaScript

7.5. Código ActionScript que implementa Cairngorm

Una vez "compilada" la librería Cairngorm, sólo hay que hacer uso de ella de forma totalmente transparente, como se hacía en la versión Flex. En este caso, el código que controla la interacción del usuario con el cliente, estará escrito en JavaScript, utilizando Base.js para la creación de clases y herencia y heredando los componentes necesarios de la librería Cairngorm.

Además, se cumplió con creces el objetivo principal, la transparencia en la migración, pues migrar eventos y comandos de Flex a HTML5 se convirtió en una tarea casi automática.

Se entrará en más detalle de esta sección en el próximo capítulo, pero a modo de ejemplo, se muestra a continuación la similitud entre un evento y un comando en Flex y en JavaScript, utilizando cada uno su correspondiente framework Cairngorm:

A continuación se muestra el evento **ExerciseEvent** en Flex:

```
1  public class ExerciseEvent extends CairngormEvent
2  {
3    // Constructor del evento
4    public function ExerciseEvent(type:String, exercise:ExerciseVO = null,
5                                  report:ExerciseReportVO = null,
6                                  score:ExerciseScoreVO = null)
7    {
8      super(type);
9      this.exercise = exercise;
10     this.report=report;
11     this.score=score;
12   }
13
14   // Variables static del evento
15   public static const EXERCISE_SELECTED:String = "exerciseSelected";
16   public static const GET_RECORDABLE_EXERCISES:String = "getRecordableExercises";
17 }
```

Y su equivalente JavaScript:

```
1  var ExerciseEvent = Cairngorm.Event.extend(
2  {
3    // Constructor del evento
4    init : function ( type, exercise, report, score )
5    {
6      this._super(type, {
7        "exercise" : exercise,
8        "report" : report,
```

```
      "score" : score
    });
  }
});

// Variables static del evento
ExerciseEvent.EXERCISE_SELECTED = "exerciseSelected";
ExerciseEvent.GET_RECORDABLE_EXERCISES = "getRecordableExercises";
```

Siguiendo en la misma línea, a continuación se muestra el comando *GetRecordable-ExercisesCommand* que es el encargado de obtener los ejercicios del servicio. Primero su código en Flex:

```
public class GetRecordableExercisesCommand implements ICommand, IResponder
{
  // Ejecuta el comando
  public function execute(event:CairngormEvent):void
  {
    new ExerciseDelegate(this).getRecordableExercises();
  }

  // Atiende el resultado exitoso
  public function result(data:Object):void
  {
    // Manage response data
  }

  // Notificar en caso de error
  public function fault(info:Object):void
  {
    CustomAlert.error(ResourceManager.getInstance().getString('myResources',
                'ERROR_WHILE_RETRIEVING_EXERCISES'));
    trace(ObjectUtil.toString(info));
  }
}
```

Y su equivalente en JavaScript:

```
var GetRecordableExercisesCommand = Cairngorm.Command.extend(
{
  // Ejecuta el comando
  execute : function ()
  {
    BP.PracticeDelegate.getRecordableExercises(this);
  },

  // Atiende el resultado exitoso
  onResult : function ( response )
  {
```

```
12      // Manage response data
13    },
14
15    // Notificar en caso de error
16    onFault : function ()
17    {
18      BP.CMS.abortLoading();
19      alert("{{$ERROR_WHILE_RETRIEVING_EXERCISES}}");
20    }
21  });
```

Como ya se ha mencionado, se entrará en detalle en el próximo capítulo de la implementación exacta de todo Babelium Project. La intención de este capítulo era servir de introducción y ofrecer equivalencias de migración entre Flex y HTML5.

Capítulo 8

Migración de Babelium Project

Contents

El presente capítulo mostrará la serie de iteraciones llevada a cabo para llevar Babelium Project desde los requisitos y objetivos del proyecto, hasta el resultado completo del proyecto con la web completamente operativa en HTML5.

Cada sección a continuación hará referencia a una iteración en la migración del proyecto, todas ellas ordenadas cronológicamente. Cada sección contendrá, una planificación inicial de la iteración, seguido de la explicación de las tareas desarrolladas a lo largo de ésta, con sus correspondientes resultados.

8.1. Prueba de concepto

Esta primera iteración se realizó con el principal objetivo de comprobar y demostrar que era factible migrar Babelium Project a HTML5 con el mismo diseño y funcionalidades.

8.1.1. Prototipo de la interfaz en HTML5

Siguiendo las buenas prácticas y las equivalencias definidas en la sección 7.2, no fue difícil crear una estructura para la página. Además, mientras se documentaba la sección mencionada, se hicieron pruebas con los Flexible Box Model y con los nuevos contenedores semánticos de HTML5, por lo que el prototipo de la interfaz estaba prácticamente terminado. Se comenzó en marzo del 2011 y se fue desarrollando a medida que investigaba sobre las equivalencias entre Flex y HTML5.

El resultado del primer prototipo constaba de la cabecera completa, incluyendo la barra de navegación y la de búsqueda, un *loader* para mostrar mientas se cargaban las secciones, y secciones sin contenido (mostrando un 404), permitiendo navegar entre secciones (aunque éstas no mostraran contenido alguno). En la figura 8.1 puede verse la primera interfaz desarrollada para HTML5, ocupando el 100 % del ancho de la página, detalle que cambiaría en un futuro.

8.1.2. Infraestructura básica para la gestión de contenidos en el servidor

Tenía que ser algo simple y funcional, pues era solamente una versión de factibilidad de la web, sin embargo, se puso en práctica los conocimientos aprendidos durante la

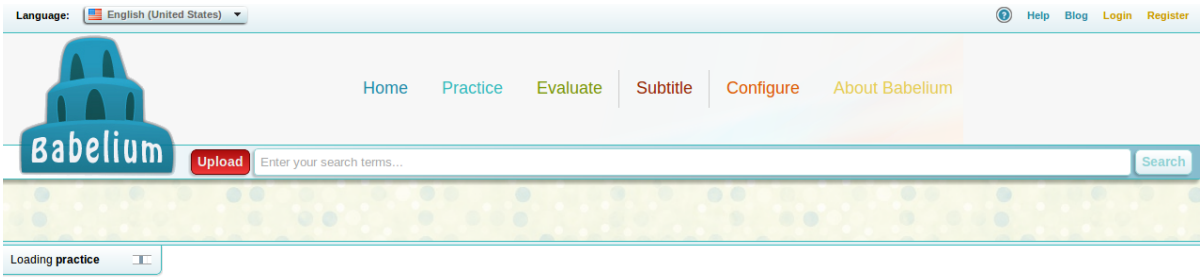


Figura 8.1: Primer prototipo de la interfaz de la web en HTML5

investigación, y dio como resultado una estructura simple, pero a su vez modularizada y ordenada para gestionar los contenidos.

Se propuso como objetivo ofrecer una navegación dinámica al usuario, es decir, ofrecer al usuario la posibilidad de navegar por la web sin tener que recargar la página al cambiar de una sección a otra. Para ello, la infraestructura contó con dos *gateways* (*bridges*) o puertas de acceso:

1. **index.php:** La única forma en la que un usuario puede acceder a la web mediante el navegador, entrando en la página index.php (modificando la configuración del Apache, se puso index.php como página por defecto). Esta página se encarga de descargar la estructura **completa** de la página cuando el usuario entra en ella, incluidos los scripts JavaScript y las hojas de estilo necesarias.

2. **modules.php:** La segunda puerta de acceso, que se encarga de ofrecer **únicamente** el contenido de las secciones que recibe por parámetro. Esta puerta de acceso es la utilizada para cargar contenido nuevo dinámicamente, sin que el usuario tenga que recargar la página.

La siguiente imagen refleja mejor el funcionamiento de las puertas de acceso:

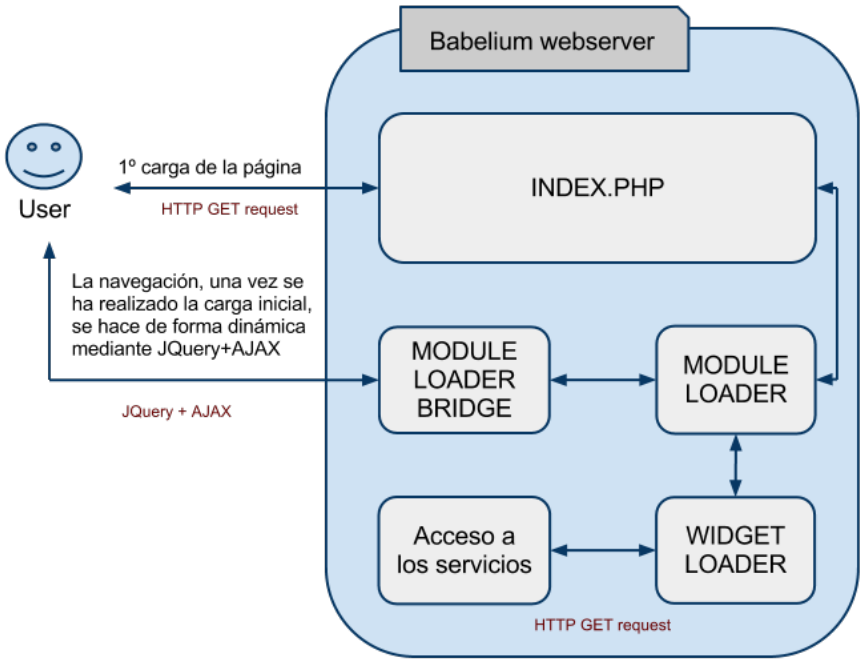


Figura 8.2: Gateways de la infraestructura inicial de Babelium Project en HTML5

Como se observa en la figura 8.2, el usuario accede a la página web mediante el *gateway* principal, solicitando el módulo deseado mediante URL, por ejemplo `/index.php?module=home`, acto seguido se carga la estructura principal de la página mediante 2 elementos esenciales, el **ModuleLoader** y el **WidgetLoader**.

ModuleLoader. Es el *loader* encargado de cargar el contenido de los módulos que se especifiquen por URL. Babelium Project está compuesto por los siguientes módulos: home, practice, evaluate, configuration, account y auth; básicamente, los enlaces disponibles en el componente de navegación principal de la web, como se veía en la figura 8.1.

WidgetLoader. Este *loader* es el encargado de buscar los widgets asociados al módulo que se desea cargar, y cargarlos con la información actualizada de la base de datos.

Una vez ModuleLoader y WidgetLoader obtienen el contenido necesario, **index.php** devuelve al navegador la estructura completa de la página, incluyendo todos los componentes JavaScript (*loader.js*) y hojas de estilo que va a necesitar para que la página funcione.

Una vez terminada la carga, la librería *loader.js* (explicada en el siguiente apartado) será la encargada de actualizar el contenido de la web dinámicamente haciendo peticiones al segundo *gateway*, **modules.php**, y actualizando el contenido de la página con los resultados.

Aunque pueda parecer un poco confuso al inicio, no se entrará mucho en el detalle del funcionamiento ahora, pues la siguiente iteración (sección 8.2) está completamente centrada en el desarrollo de una sólida estructura modularizable y fácilmente escalable.

8.1.3. Loader.js

Se creó una librería en JavaScript pequeña y muy resumida (apenas 20 líneas de código) solamente con la finalidad de comprobar la factibilidad de ofrecer a la web navegación dinámica.

Esta librería era la encargada de, cuando un usuario pulsase en un enlace de navegación, realizar las siguientes tareas:

1. Mostrar aviso de "loading..."
2. Contactar con el segundo *gateway*, **modules.php** enviándole como parámetro el módulo al que desea cambiar, ejemplo: `/modules.php?module=practice` y obtener el contenido del módulo.
3. En caso de éxito en la consulta, actualizar el contenido de la sección principal.
4. En caso de error en la consulta, mostrar una alerta avisando al usuario de que se había producido un error.

Las consultas las realizaba mediante JQuery y AJAX, y actualizaba el contenido de la sección principal gracias a la inserción de contenidos de JQuery:

```
1  var query_string = "/modules.php?module=" + location;
2
3  // Mostrar Loading
4  $("#maincontent > aside#loader").slideDown(500);
5
6  // Remover contenidos actuales del contenedor principal
7  $("#maincontent > section").remove();
8
9  // Obtener nuevos contenidos mediante AJAX
10 $.getJSON(query_string, function(data)
11 {
```

```
12   // Parsear los contenidos recibidos
13   var content = $(data.content);
14   // Insertar en el contenedor principal los contenidos recibidos
15   content.appendTo("#maincontent");
16   // Ocultar loading
17   $("#maincontent > aside#loader").slideUp(500);
18 });
```

Siendo **location** el módulo al que se quiere acceder, y **#maincontent** el contenedor que contiene la sección principal de la web. La labor del script es eliminar las secciones actuales dentro del contenedor principal, obtener las nuevas mediante AJAX, y sustituirlas. Como nota aclarativa, $ es el signo que identifica un comando de JQuery.

8.1.4. Plantillas Smarty

Para la gestión de contenidos en el servidor, además de la infraestructura básica antes mencionada, se utilizó Smarty. Smarty es un motor basado en plantillas, y las plantillas son, bloques de código HTML (en nuestro caso, pero pueden ser cualquier tipo de texto) que varían su contenido en función del usuario o de otros factores externos. Por ejemplo, a continuación se muestra el ejemplo de una plantilla HTML:

```
1 <html>
2   <head>
3     <title>{$title}</title>
4   </head>
5   <body>
6     Hello, {$name}!
7   </body>
8 </html>
```

La utilidad de las plantillas, no es otra que crear contenido genérico, independiente del usuario que la visita o independientemente de la base de datos, y actualizar los datos mediante un motor de plantillas posteriormente. Por ejemplo, en el ejemplo anterior veíamos una plantilla con 2 variables definidas **$title** y **$name**, gracias a Smarty se puede mostrar esa plantilla cargando datos distintos en función del usuario (suponiendo *hello.tpl* el nombre con el que hemos guardado la plantilla anterior):

```
1 $smarty->assign('title', 'Prueba de proyecto');
2 $smarty->assign('name', 'Imanol');
3 $smarty->display('hello.tpl');
```

Se mostrará en pantalla una página formada por la plantilla anterior sustituyendo las variables por los datos proporcionados, dando como resultado un "Hello, Imanol!".

Se utilizó Smarty para separar los módulos y widgets en plantillas separadas, y cargar cada una de ellas en función de las peticiones. Por ejemplo, el primer prototipo se utilizaron las plantillas para separar al web en trozos y cargarlos dinámicamente:

- La cabecera no visible de la página (el *head* de HTML) que contiene información de todas las hojas de estilo y scripts que se tiene que descargar.
- La cabecera visible de la página, conteniendo el menú de usuario (cuenta, login, logout,..), el logo, el menú de navegación y la barra buscadora.
- Una serie de plantillas que se cargan dinámicamente en función de la sección en la que se encuentra el usuario
- Una plantilla con el pie de página

8.2. Infraestructura y gestión de sesiones

8.2.1. Infraestructura de la aplicación

Se ha ido mencionando en capítulos anteriores la infraestructura de la aplicación. Pero para entenderla mejor, esta sección se centrará completamente en ella, desglosándola desde lo más básico hasta su total complejidad.

Vamos a empezar descomponiendo poco a poco el funcionamiento interno de Babelium Project HTML5. Como cualquier aplicación, está basada en una arquitectura de 3 niveles, como se ve en la figura 8.3.

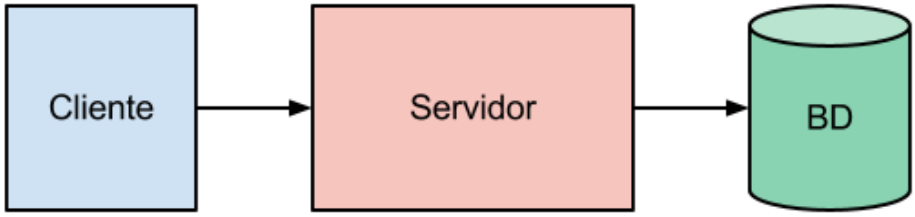


Figura 8.3: Arquitectura de 3 niveles

Sin embargo, en nuestro caso, se ha creado una aplicación orientada a servicios, con lo que el cliente dispone de 3 *gateways* diferentes en el servidor, en función del tipo de servicio o información que requiera. La figura 8.4 lo muestra claramente:

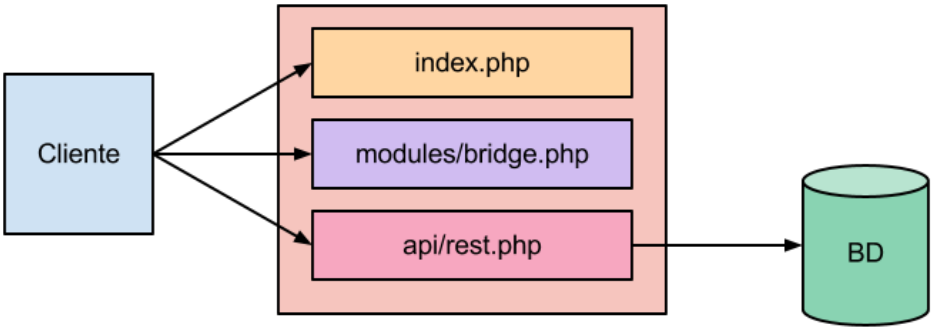


Figura 8.4: Arquitectura de 3 niveles con 3 gateways

index.php es el gateway principal de la aplicación. Se encarga de cargar por primera vez la estructura completa de la aplicación en el cliente.

modules/bridge.php es un gateway para carga dinámica de módulos y/o widgets. Los módulos principales son: Home, Practice, Evaluation, Subtitle, Help, About, Account, Upload y Auth. Este gateway SIEMPRE devuelve contenido HTML ya estilizado utilizando plantillas y llamadas a la API (siguiente gatway) para obtener el contenido.

api/rest.php es el gateway de la API de la aplicación. Es el único gateway con acceso a la base de datos y devuelve los objetos codificados en JSON.

Por lo general, el cliente no va a acceder directamente al gateway de la API (aunque podría), así que definimos la estructura de la siguiente forma, indicando una distinción de 2 niveles dentro del servidor: nivel de negocio y nivel de api:

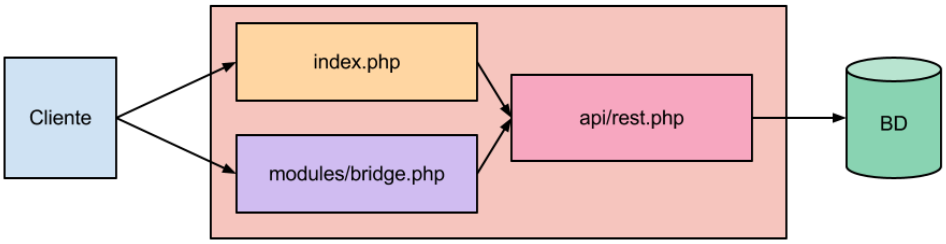


Figura 8.5: Arquitectura de 3 niveles con distinción de 2 niveles en el servidor

De la misma forma, el cliente se divide en varios módulos, y nunca llamará primero al gateway de los módulos. Siempre se realizará primero una única llamada al gateway *index.php* para descargar la estructura completa y las librerías JavaScript necesarias, y después mediante AJAX el cliente irá cargando dinámicamente el contenido de la página; secciones y/o módulos utilizando el gateway prefijado para dicho uso (*modules/bridge.php*):

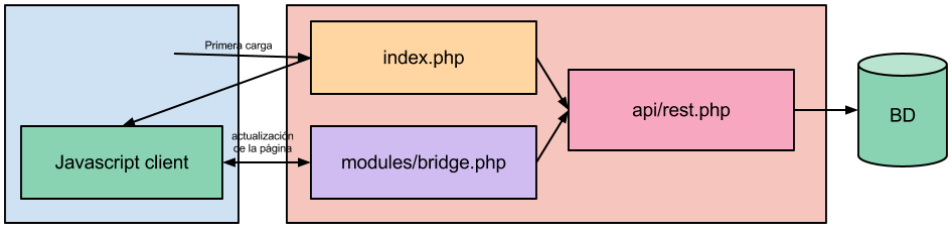


Figura 8.6: Arquitectura de 3 niveles con distinción con carga dinámica de contenidos

Ahora que ya sabemos la estructura básica de la aplicación, las siguientes 2 subsecciones desglosarán su funcionamiento en cliente y servidor.

Arquitectura del cliente

El cliente, una vez cargado *index.php* y descargadas todas las librerías JavaScript necesarias, hace uso de dichas librerías, que son las siguientes:

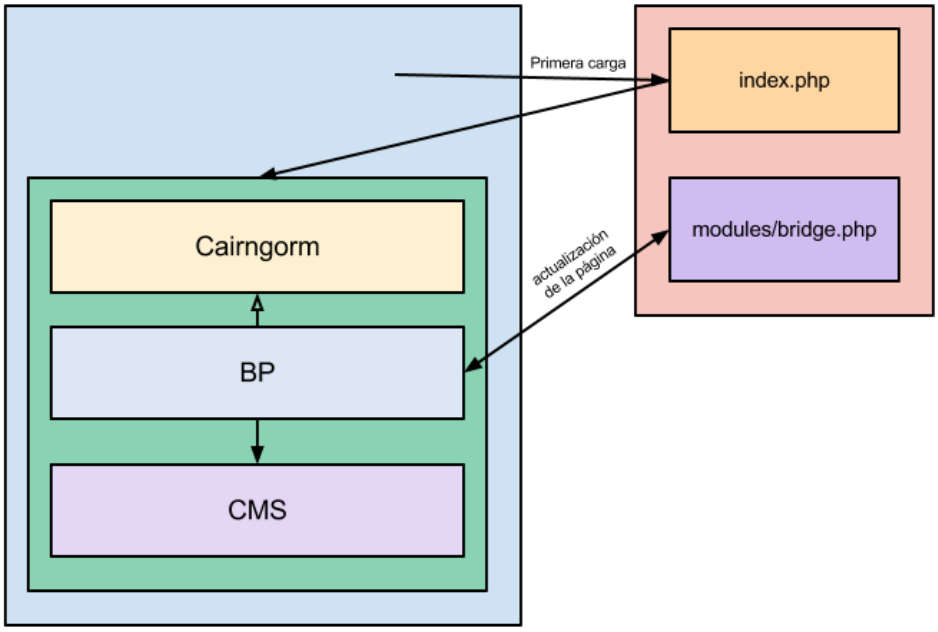


Figura 8.7: Arquitectura de 3 niveles con distinción con carga dinámica de contenidos

Cairngorm es una librería que simula el funcionamiento de Cairngorm MVC Framework de Flex. Implementa todos los componentes principales de dicha aplicación, aunque por ahora, sólo implementa las funcionalidades que Babelium Project necesita.

BP es una serie de objetos propios de Babelium Project que implementan el framework de Cairngorm.

CMS o *Content Management System* es otro objeto que se encarga de facilitar a BP la inserción dinámica de contenidos mediante efectos en la página web.

Profundizando más, recordamos de la sección 7.4, Cairngorm es un framework que ofrece el patrón MVC que funciona en base a: Eventos, Comandos, y Delegados. En la figura 8.8 se puede observar su desglose dentro del cliente.

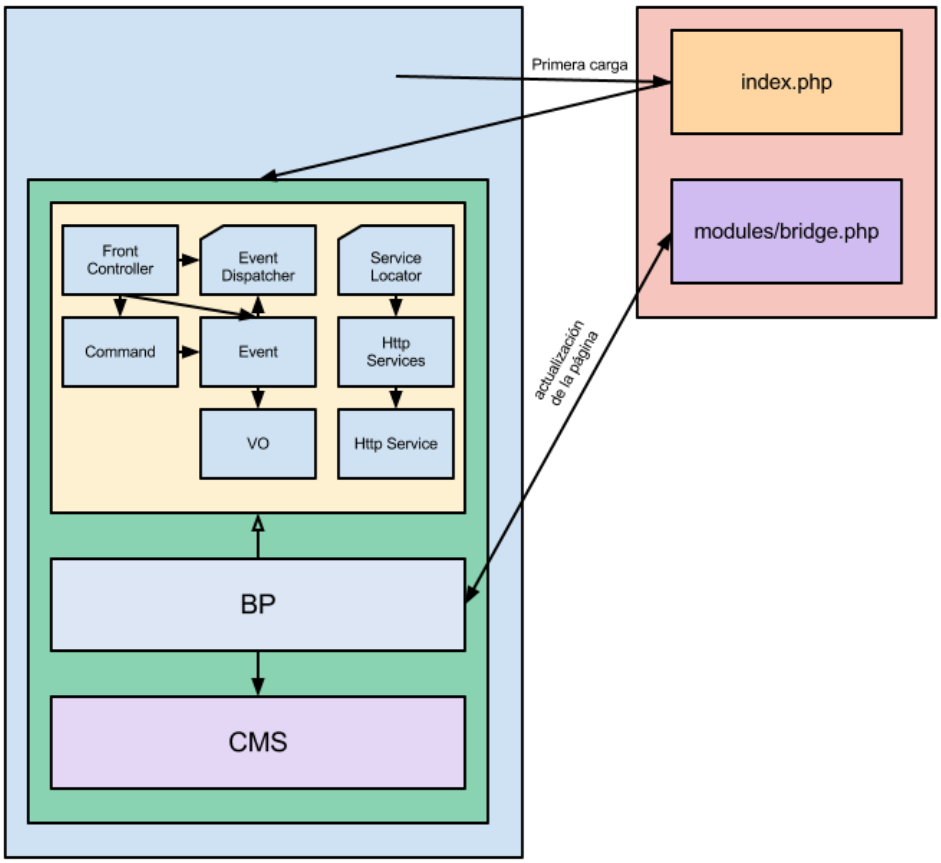


Figura 8.8: Desglose de Cairngorm en el cliente

Sin embargo, como ya se mencionó anteriormente, esta estructura de Cairngorm es solamente un framework, una librería para su posterior uso. No tiene ningún uso por si misma. Es por ello que creamos el objeto/librería **BP**, que es una implementación de la arquitectura Cairngorm para BabeliumProject.

En la figura 8.9 se puede observar una lista de objetos que componen dicha implementación. Como se muestra en la imagen, se han creado gran cantidad de eventos, comandos, servicios y delegados para implementar el framework de Cairngorm:

Events: conjunto de eventos donde cada uno, extiende a *Cairngorm.Event*. Por ejemplo, **LoginEvent**, el cual tiene 2 tipos internos PROCESS_LOGIN y SIGN_OUT. El

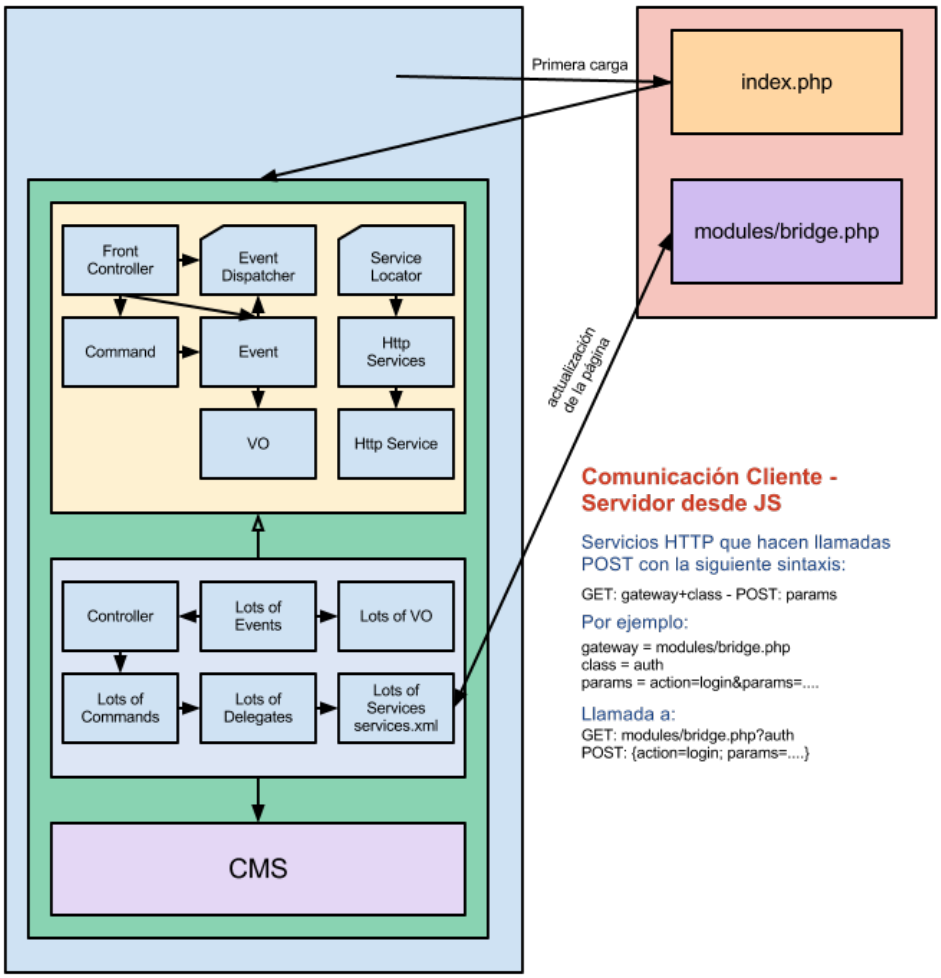


Figura 8.9: Desglose de BP en el cliente

constructor de LoginEvent requiere un parámetro adicional (aparte del tipo de evento) que es un `LoginVO` (un objeto con datos de usuario al que pretende identificar).

Commands: comandos que extienden *Cairngorm.Command*. Por ejemplo **ProcessLoginCommand** y **SignOutCommand**, los cuales implementan además el patrón Responder que se explica más adelante. El comando ProcessLoginCommand además, recoge el objeto `LoginVO` que el evento LoginEvent almacena para luego intentar identificar al usuario con dichos datos.

VOs: Son objetos simples que contienen únicamente datos. Un ejemplo es el objeto `LoginVO` que se ha mencionado anteriormente, que contiene únicamente el nom-

bre y contraseña del usuario que quiere identificarse. Los VO **se representan** mediante JSON en JavaScript, por ejemplo, el siguiente trozo de código sería una representación en JSON del objeto LoginVO de Flex:

Controller: objeto que extiende al *Cairngorm.FrontController* y que además añade a la tabla de correspondencias todos los eventos -¿ comandos mencionados arriba. Cuando un evento propio se dispara, este controlador ejecuta nuestros comandos asociados al mismo.

Services: se ha creado un archivo de configuración (*services.xml*) el cual contiene 2 tipos de objetos: gateways y servicios. Los gateways tienen una ID, un objetivo, un tipo y un método, por ejemplo:

Mientras que los servicios tienen una ID, un destino que es a su vez una referencia a una ID de gateway, y una clase que viene a ser el módulo de destino, por ejemplo:

Al cargar la página, el script loader que se encarga de inicializar toda la infraestructura de Cairngorm y BP, por cada una de las entradas del fichero **services.xml**, crea un objeto HTTPService con dichos datos, por ejemplo para el ejemplo puesto arriba se crearía un servicio que lanzaría peticiones a "modules/bridge.php?home" y este servicio sería identificado por `homeMOD` en el HTTPServices, ID por el que ServiceLocator de Cairngorm sería capaz de encontrarlo. A continuación se muestra un fragmento breve de **services.xml** para comprender mejor lo anterior:

```
1 <services>
2   <gateway id="MODULES" target="modules/bridge.php?" type="http" method="post" />
3
4   <service id="home" destination="MODULES" class="home" />
5 </services>
```

Delegates: los delegados no son propios de Cairngorm, pero son propios de una buena implementación del patrón Delegate para abstraer la extracción de información externa. Se han creado delegados para cada servicio HTTP disponible. Por ejemplo, para acceder a los servicios HTTP de **auth**, se ha creado el delegado AuthDelegate, el cual contiene dos métodos: processLogin y signOut. Ambos métodos reciben como parámetro un objeto de tipo Responder. Cada delegate tiene asociado un ID de servicio. Cuando se llama a alguno de sus métodos, obtienen del ServiceLocator el servicio asociado a dicha ID, y asignan el comando que llamó al delegate como Responder. El delegado lanza la llamada al servicio, y se encarga de que el comando reciba la respuesta en el método onResult (patrón responder).

Para terminar de digerir mejor todo lo explicado anteriormente, se muestra en la figura 8.10 un diagrama de secuencia de cómo consigue procesarse una acción de login.

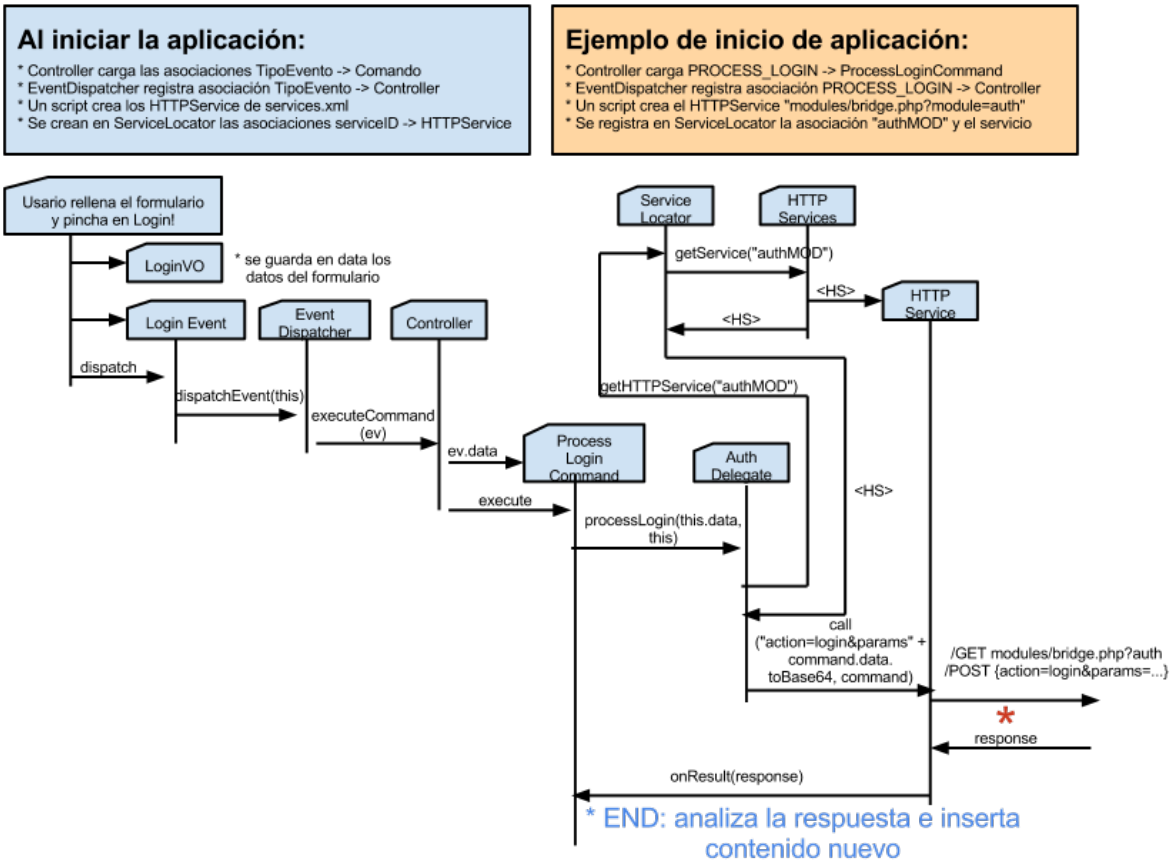


Figura 8.10: Diagrama de secuencia de la acción login en el cliente

Arquitectura del servidor

La arquitectura en la parte del servidor, continúa la propuesta por la primera iteración (sección 8.1). A continuación se explicará su funcionamiento más detalladamente. Primero, se muestra en la figura 8.11 un diagrama del funcionamiento de la arquitectura en el servidor, y a continuación se procederá a explicar todos y cada uno de sus componentes.

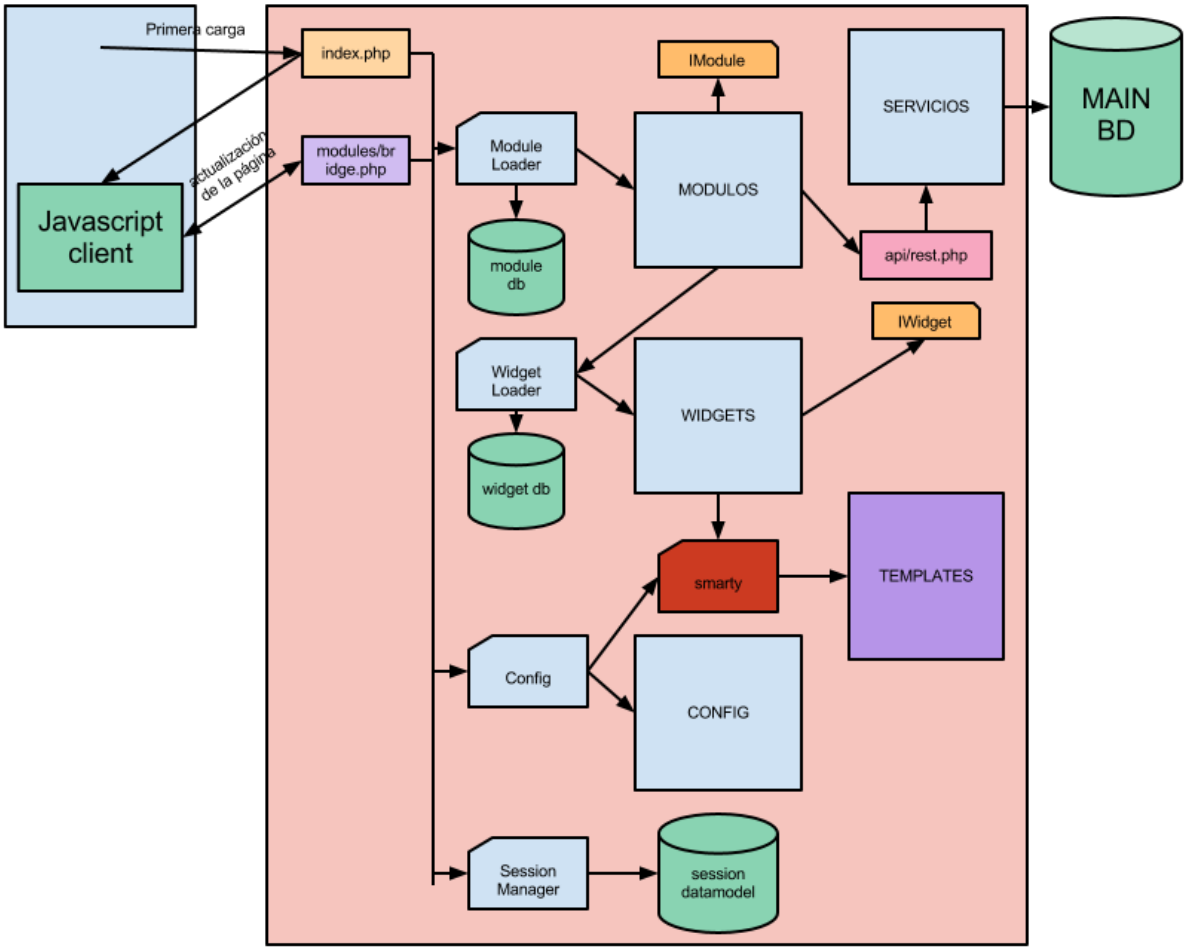


Figura 8.11: Arquitectura de la aplicación en el lado servidor

La aplicación contiene, como se ha explicado anteriormente, 2 gateways, que acceden a la misma infraestructura, con la diferencia de que **index.php** obtiene información adicional y la devuelven de forma distinta. Estos gateways interactúan principalmente

con 4 **instancias únicas**[1] que hacen de núcleo (o *core*) de la aplicación: ModuleLoader, WidgetLoader, Config y SessionManager.

ModuleLoader. Más que una instancia, es una clase que contiene únicamente un método estático (`loadModule`) y una tabla hash con correspondencias `moduleName -> moduleClass`. La funcionalidad de esta instancia, es invocar la carga de un módulo u otro en función de la petición del cliente.

WidgetLoader. El funcionamiento es casi calcado al de `ModuleLoader`, con la diferencia de que ésta es la encargada de invocar la carga de distintos widgets en función del módulo y el estado de éste.

Config. Es una instancia única con toda la configuración de la aplicación.

SessionManager. es otra instancia única que controla la sesión del usuario. Además, implementa en su interior un DataModel. Tiene métodos getVar y setVar para asignar u obtener variables dentro de la sesión del usuario (se mantienen de una llamada a otra) utilizando las sesiones de PHP para ello.

Modulos y widgets Pese a que el funcionamiento del ModuleLoader y WidgetLoader desde un punto de vista de programación, es prácticamente el mismo, se decició distinguirlo en dos componentes distintos por mera semántica, escalabilidad y reusabilidad, bajo las siguientes definiciones.

- *Un módulo es una sección que puede estar en distintos `estados`, y cada estado obtiene información de la base de datos y está compuesto por uno o más widgets*
- *Un widget es una plantilla HTML+CSS para representar la información de la base de datos*

Al distinguirlos de esta manera, se mantienen los widgets como plantillas abstractas reutilizables, es decir, distintos módulos pueden reutilizar el mismo widget (para mostrar información de un vídeo, por ejemplo).

Fuentes del núcleo de la aplicación Las cuatro instancias únicas mencionadas anteriormente, contienen cada una una base de datos propia con la información que necesita para funcionar.

[1] Clases de las cuales existe una **única** instanciación para toda la ejecución del programa

ModuleLoader. Contiene una base de datos (en forma de tabla hash) de módulos disponibles y un módulo de error (404) por si se recibe una petición para un módulo inexistente.

WidgetLoader. Al igual que el ModuleLoader, contiene una base de datos (en forma de tabla hash) que contiene la lista de widgets disponibles y un widget de error 404.

Config. Esta instancia guarda la configuración en un archivo XML que se carga al inicio de cada ejecución.

SessionManager. Esta instancia genera una sesión por cada cliente que se conecta, y dentro de cada sesión, guarda variables propias de cada uno de ellos; dicho de otra forma, dentro de las sesiones de PHP guarda un *DataModel* (modelo de datos) que representa al cliente e información sobre la visita a la web del mismo.

Funcionamiento de la aplicación en el lado servidor Para comprender mejor su funcionamiento, vamos a mostrarlo mediante un ejemplo, y más concretamente, siguiendo el ejemplo del lado cliente: la autenticación de un usuario. En el servidor disponemos de los siguientes objetos:

- Instancias únicas mencionadas arriba como *core* de la aplicación.
- Un módulo llamado Auth con dos estados posibles, **login** y **logout**, y su correspondiente entrada en la base de datos de módulos.
- Por cada estado tendremos: una conexión a los servicios para procesar datos, un widget que lo representa (**LoggedIn** y **LoggedOut**) y sus correspondientes entradas en la base de datos de Widgets.

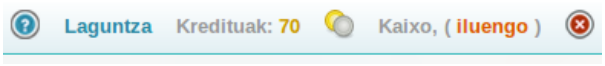


Figura 8.12: Widget LoggedIn

Figura 8.13: Widget LoggedOut

- Plantillas HTML+CSS3 para representar dichos widgets
- Servicios en la API con la posibilidad de procesar una petición de login y logout

Y la secuencia de acciones que se realizarían en el servidor una vez recibida la petición de login, obviando la carga de configuración y la creación de sesiones, sería la siguiente:

```
1 Request:
2 /GET modules/bridge.php?auth
3 /POST {action: 'login', params: 'user=iluengo&pass=iluengopass'}
```

1. El *gateway* invocaría el **ModuleLoader** para cargar el módulo `auth` enviándole también los parámetros recibidos en el POST.
2. **ModuleLoader** recibiría el módulo a cargar (auth), la acción a realizar o estado en el que se encuentra el módulo (login) y los parámetros necesarios para poder procesar dicha acción (user=iluengo&pass=iluengopass) y lanzaría la carga del módulo Auth (habiendo comprobado que existe en la base de datos) con dichos parámetros.
3. El módulo Auth, al estar en el estado de login, descodificaría los parámetros (user y pass) y lanzaría una llamada a la API para intentar identificar el usuario. En caso de identificación fallida, devolvería un mensaje de error. En caso contrario seguiría en el siguiente punto.
4. Con la información recibida de la API (información de la cuenta de usuario, como los cŕeditos que tiene), el módulo Auth invocaría el **WidgetLoader** pidiéndole que cargue el widget **LoggedIn** y enviádole como parámetro la información de usuario recibida.
5. **WidgetLoader** comprobaría la existencia del widget LoggedIn y le enviaría la información del usuario al widget.
6. El widget **LoggedIn**, mediante el gestor de plantillas Smarty, invocaría la carga de la plantilla HTML+CSS *LoggedIn.tpl* sustituyendo las variables existentes en la plantilla (nombre y créditos del usuario, por ejemplo) por la información del usuario recibida.
7. Las plantillas procesadas con Smarty se devuelven al cliente y el CMS (Content Management System) en JavaScript se encargaría de actualizar dinámicamente el contenido de la página.

La figura 8.14 refleja los anteriores puntos.

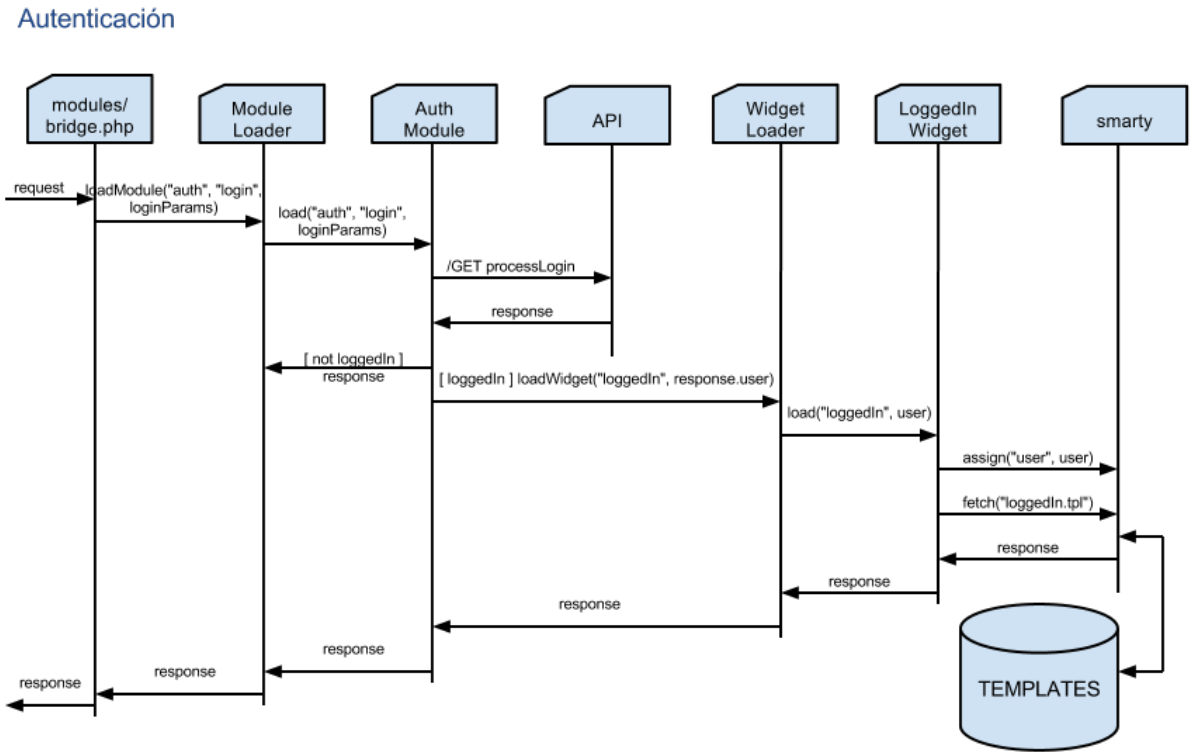


Figura 8.14: Diagrama de secuencia de la acción login en el servidor

8.2.2. Content Management System

Para el cliente, además, se diseño un gestor de contenidos (en ingles CMS o *Content Management System*). Una clase en JavaScript preparada para entender la estructura de la web y cambiar/insertar contenido dinámicamente. La clase, `BP.CMS`, contiene, entre otros, los siguientes métodos:

prepareMainContent(text, callback): método principal para ocultar la sección actual

1. Oculta TODO el contenido de la sección actual, utilizando efectos gráficos de JQuery
2. Muestra el mensaje de loading con el texto `text`
3. Llama a la función `callback` cuando ha terminado de ocultar todo

prepareExerciseView(callback): sólo usable en la sección *practice* o *evaluate*

1. Oculta (si existiera) la información del ejercicio que se está realizando o evaluando
2. Muestra ventana de loading
3. Llama a la función callback

innerMainContent(data): se utiliza después de prepareMainContent

1. Oculta la ventana de loading
2. Inserta, mediante efectos de JQuery, el nuevo contenido de la sección principal
3. Refresca los componentes especiales (paginations, datatables,..) si los hubiera

innerExerciseView(data): se utiliza despues de prepareExerciseView

1. Oculta la ventana de loading
2. Muestra la información del nuevo ejercicio o evaluación
3. Refresca los componentes especiales (paginations, datatables,..) si los hubiera

Además, dado que las inserciones de contenido y la captura de información se realiza mediante AJAX, se creó este gestor para evitar colapso de peticiones. Es decir, el gestor impide la obtención de contenido nuevo si aún se está atendiendo a otra petición, por ejemplo: si el usuario pincha en un enlace, y acto seguido en otro (o si pulsa en muchos enlaces en un breve lapso de tiempo) el gestor rechazará el resto de peticiones hasta que termine la primera.

8.3. Protocolos de comunicación, navegación y API

La siguiente iteración del proyecto se llevo a cabo para definir unos protocolos de comunicación robustos y para actualizar los servicios de Babelium Project a la nueva versión de la API.

8.3.1. Protocolo de comunicación con la API

Como ya mencioné anteriormente, **Inko Perurena** que trabaja como investigador para el grupo de investigación GHyM, me proporcionó ayuda con algunas tareas, ya que eran comunes a la versión HTML5 y la versión Flex. Es por ello, que Inko ya había desarrollado el gateway de la API para la versión de Flex para poder usar los servicios de forma externa. Mi trabajo en esta parte, sólo consistió en actualizar la API de la versión de la versión HTML5 a la versión que estaba disponible para Flex e implementar el protocolo de conexión.

El protocolo de conexión a la API está basado en tickets. Antes de hacer peticiones a la API para obtener datos de la aplicación, el cliente tiene que pedir un ticket para poder acceder a ella. Para ello, se implementó en el cliente JavaScript un script (**BP.Services**) que siguiera los siguientes pasos necesarios:

1. Obtener el ticket de comunicación (también llamado *token*)

   a) Obtener la ID de la sesion del cliente [28]

```
this.getSessionID = (function ()
{
  var results = document.cookie.match('(^|;) ?'
        + 'PHPSESSID' + '=([^;]*)(;|$)');
  if (results)
    return (unescape(results[2]));
  else
    return null;
  })();
```

   b) Generar un uuid (*unique user id*) [29]

```
this.getUUID = (function ()
{
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g,
  function(c) {
    var r = Math.random()*16 | 0, v = c == 'x'? r : (r & 0x3 | 0x8);
    return v.toString(16);
```

```
  }).toUpperCase();
})();
```

c) Obtener el token de comunicación lanzando una petición HTTP al gateway de la API

```
/GET http://<domain>/api/rest?getCommunicationToken
/POST
{
  method: getCommunicationToken
  header: {session: <phpsessid>, secretKey: md5(<phpsessid>),
      uuid: <gen_uuid>}
}
```

2. Lanzar peticiones a la API

a) Obtener el método y parámetros requeridos de la API

b) Generar una secuencia de 6 caracteres aleatorios

```
this.createRandomSalt = function(){
  var randomizer = '';
  var charsGenerated = 0;
  while (charsGenerated < 6){
    randomizer = randomizer +
            (Math.floor(Math.random() * 16)).toString(16);
    charsGenerated++;
  }
  return randomizer !== this.lastRandomizer ? (randomizer)
            : (createRandomSalt());
};
```

c) Generar un token encriptado de comunicación, siendo `current_method` el método de la API solicitado, `commToken` el token recibido en el primer paso, `statToken` una cadena de caracteres invariable y compartida tanto en el cliente como en el servidor, y `salt` los 6 caracteres aleatorios generados en el paso inmediatamente anterior:

```
salt = generate_6char_random_salt();
current_comm_token = salt + Sha1(current_method + ':' +
    commToken + ':'+ statToken + ':' + salt);
```

```
this.generateToken = function (method) {
  var salt = this.createRandomSalt();
```

```
3   var t = Sha1.hash(method + ":" + this.commToken + ":" +
4             this.statToken + ":" + salt);
5   var s = salt + t;
6   return s;
7 };
```

d) Lanzar peticiones HTTP a la API con el siguiente protocolo:

```
1   /GET
2   http://<domain>/api/rest?<methodName>
3
4   /POST
5   {
6     method: <methodName>,
7     paramaters: { <param_1>: <value_1>, <param_n>: <value_n> },
8     header: { uuid: <gen_uuid>, token: <current_comm_token>,
          session_id: <phpsessid> }
9   }
```

8.3.2. Protocolos de comunicación para la gestión de contenidos

Recordamos que la aplicación en el servidor contaba con 2 *gateways* principales, **index.php** para la primera carga y **modules/bridge.php** para la obtención dinámica de contenidos nuevos.

Ambos protocolos funcionan sobre HTTP, como era de imaginar, y ambos aceptan 4 parámetros posibles:

module. Haciendo referencia al nombre del módulo que desea cargar. Ejemplo: módulo *evaluate* devolvería la página principal del módulo

action. Se refiere a la acción o estado en el que se encuentra el módulo. Ejemplo: módulo *evaluate* en el estado (acción) *pending* devolvería la lista de ejercicios pendientes de evaluación.

params. Parámetros necesarios para procesar el estado en el que se encuentra el módulo. Ejemplo: módulo *evaluate* en el estado (acción) *pending* con el parámetro *339* devolvería el ejercicio cuya ID es 339 y el panel para valorarlo.

state. Este parámetro es un flag que puede obtener el valor *min* para hacer que devuelva sólo una versión reducida del módulo (por ejemplo, sólo la información del ejercicio, sin devolver la lista de todos los ejercicios). Aunque parezca un poco ambiguo, este

parámetro se utiliza para devolver información muy concreta dentro de la misma sección, es decir, si estamos evaluando un vídeo, y queremos evaluar otro, el cliente sólo necesita recibir la información del nuevo ejercicio, no necesita recibir de nuevo la lista completa de ejercicios. Esté parámetro sólo se utiliza mediante el gateway **modules/bridge.php**.

Protocolo de index.php

Recibe los parámetros arriba mencionados mediante el método GET para cargar la página web en un módulo y/o estado concreto:

```
/GET /?module=<module>&action=<action>&params=<params>
```

Y para el ejemplo mencionado arriba, la llamada HTTP que devolvería la página **completa** del módulo evaluación, con la estructura completa de la página, las librerías JavaScript y las hojas de estilo necesarias, la lista de ejercicios pendientes de evaluación, y el ejercicio 339 para evaluar, sería la siguiente:

```
/GET /?module=evaluate&action=pending&params=339
```

Protocolo de modules/bridge.php

De manera similar al anterior, recibe los mismos parámetros pero combinando métodos GET y POST para mayor seguridad. Este gateway recibirá **únicamente** el contenido del módulo + estado + parámetros que pide, sin tener en cuenta el resto de la página (cabecera, menú de navegación, librerías JavaScript, etc...). Sólo recibirá contenido nuevo para sustituir el viejo, y ofrecer así una navegación ligera por la web.

```
/GET /modules/bridge.php?<module>
/POST
{
   action: <action>, params: <params>, state: <state>
}
```

La llamada a este gateway devolvería únicamente el contenido del módulo evaluación, con su lista de ejercicios pendientes y la información para evaluar el ejercicio 339:

```
/GET /modules/bridge.php?evaluate
/POST
{
  action: 'pending', params: '339'
}
```

8.3.3. Gestión de la navegación por la web

Una de la práctica más común de los usuarios que navegan por internet, es compartir enlaces, o utilizar las flechas "atrás" y "adelante" del navegador. Sin embargo, ésto entra en conflicto con la navegación dinámica por la web, ya que, al actualizar el contenido de la página sin recargarla, por defecto, no cambia la URL de la misma, y es por ello que no se podrían compartir enlaces a secciones, ni utilizar los botones de navegación del navegador.

Para solventar este problema, tanto Babelium Project en Flex como prácticamente cualquier web que utilice navegación dinámica, utilizan una técnica llamada *DeepLinking* para poder enlazar a secciones o estados concretos de la aplicación.

DeepLinking es un método que permite alterar la URL manualmente mediante los siguientes recursos:

- El fragmento *hash* de la URL, conocido como lo que va detrás del símbolo #.
- Scripts en el cliente que se encargan de leer el *hash* de la URL al cargar la página, y editarlo en cada cambio de sección.

Ejemplo de url con *DeepLinking*:

```
/GET prueba.php#/home/practice/exercises
```

Sin embargo, una de las novedades de HTML5 es el *history API*. Es una de las nuevas APIs JavaScript que implementa HTML5, ofrece mediante JavaScript acceso al *history* o historial de navegación del navegador.

Esta API ofrece dos funciones básicas, que son las que se han utilizado para crear para Babelium Project un gestor de navegación:

window.history.pushState: Funcion que sirve para añadir una entrada al historial de navegación, así como para **cambiar la URL sin recargar la página**.

```
history.pushState(<stateObj>, <title>, <newUrl>);
```

El método pushState recibe 3 parámetros: `stateObj` que es un objeto JSON que se guardará dentro del estado, el título que se guardará en el historial, y la URL que aparecerá en el historial y por la que se reemplazará la actual **sin recargar la página**, ejemplo:

```
history.pushState({module: 'home'}, "Home", "index.php?module=home");
```

window.onpopstate: Es la función opuesta a la anterior. Esta función hace de callback cuando se altera manualmente (o por script) el estado de la web, por ejemplo: cuando un usuario utiliza las flechas atrás o adelante del navegador, se llama como callback a esta función devolviendo el `stateObj` del estado anterior (si ha pulsado atrás) o posterior (si ha pulsado adelante). Ejemplo:

```
window.onpopstate = function(event)
{
  alert("module: " + event.state.module));
};
```

Ejemplo de uso:

1. Accedemos al módulo HOME, y el gestor de estados del cliente lanzará el método *pushState* registrando en el historial que hemos accedido a home, modificando la URL a `?module=home` y enviando como *stateObj* el valor {`module: 'home'`}.
2. Accedemos al módulo PRACTICE ahora, y el gestor volverá a llamar al método *pushState*, registrará en el historial que se ha accedido al módulo practice, modificará la URL a `?module=practice` y guardará en el *stateObj* el valor {`module: 'practice'`}.
3. El usuario pulsa la flecha "atrás" en el navegador (o se retrodece atrás mediante script).
4. Se lanza el método onPopState, que recibe como parámetro el estado anterior en el que se encontraba ({`module: 'home'`} y se reemplararía la URL a `?module=home` de nuevo.

5. Se encarga de actualizar la web al estado anterior; cargaría dinámicamente el módulo home

Los cuatro pasos anteriores puede representarse de la siguiente manera mediante código JavaScript, para que se entienda mejor:

```
1  window.onpopstate = function(event)
2  {
3    // event.state es el stateObj del estado anterior
4    lastModule = event.state.module;
5    // Funcion que carga el modulo anterior
6    loadModule(lastModule);
7  };
8
9  // Entramos al modulo HOME
10 window.history.pushState({module: 'home'}, 'home', '?module=home');
11 // Entramos al modulo PRACTICE
12 window.history.pushState({module: 'practice'}, 'practice', '?module=practice');
13 // Se pulsa la tecla atras del navegador
14 window.history.back();
```

`window.history.back()` hace el mismo efecto que pulsar manualmente el botón atrás en el navegador. Dispararía la función `onpopstate` que recibiría el módulo *home* como parámetro, a continuación, la función `loadModule` (que no se explicará aquí), cargaría el módulo home dinámicamente.

La ventaja de usar este método frente a utilizar DeepLinking, es que la URL cambia físicamente en cada `pushState` o `popstate`, por lo que el usuario puede actualizar la página o compartir el enlace, y al entrar en dicho enlace, **index.php** cargaría la estructura completa de la página en dicho estado. En resumen, ofrece navegación dinámica y estática al mismo tiempo.

8.4. Auth Module

El siguiente paso, antes de proseguir, consistió en implementar la gestión de usuarios: registro y autenticación. Teniendo ya la infraestructura terminada, tanto en el cliente como en el servidor, sólo quedaba realizar las implementaciones necesarias.

8.4.1. Login/Logout

Siguiendo la misma arquitectura que la aplicación Flex, para poder crear el módulo de autenticación, simplemente hay que implementar los siguientes elementos en el cliente:

- Evento LoginEvent que se disparará al pulsar el botón de login
- Comando ProcessLoginCommand que utilizará el *AuthDelegate* para procesar el login, el *Content Management System* para actualizar el contenido una vez identificado y el *StateManager* para actualizar el estado de la página.
- Registro de la asociación `LoginEvent.LOGIN ->ProcessLoginCommand` en el Controlador
- Método para procesar el Login en el AuthDelegate
- Crear el *AuthModule* con dos estados: login y logout
- Crear los widgets *LoggedIn* y *LoggedOut*
- Registro del módulo Auth y los widgets LoggedIn y LoggedOut en ModuleLoader y WidgetLoader respectivamente

En la figura 8.15 se pueden observar todos estos elementos y su interacción:

Como se puede observar, y a modo de leyenda:

- Las clases coloreadas de verde son las que han tenido que crearse para añadir el módulo
- Las clases coloreadas de azul han sido las que han tenido que ser ligeramente modificadas para añadir un registro o asociación (de evento ->comando, por ejemplo)

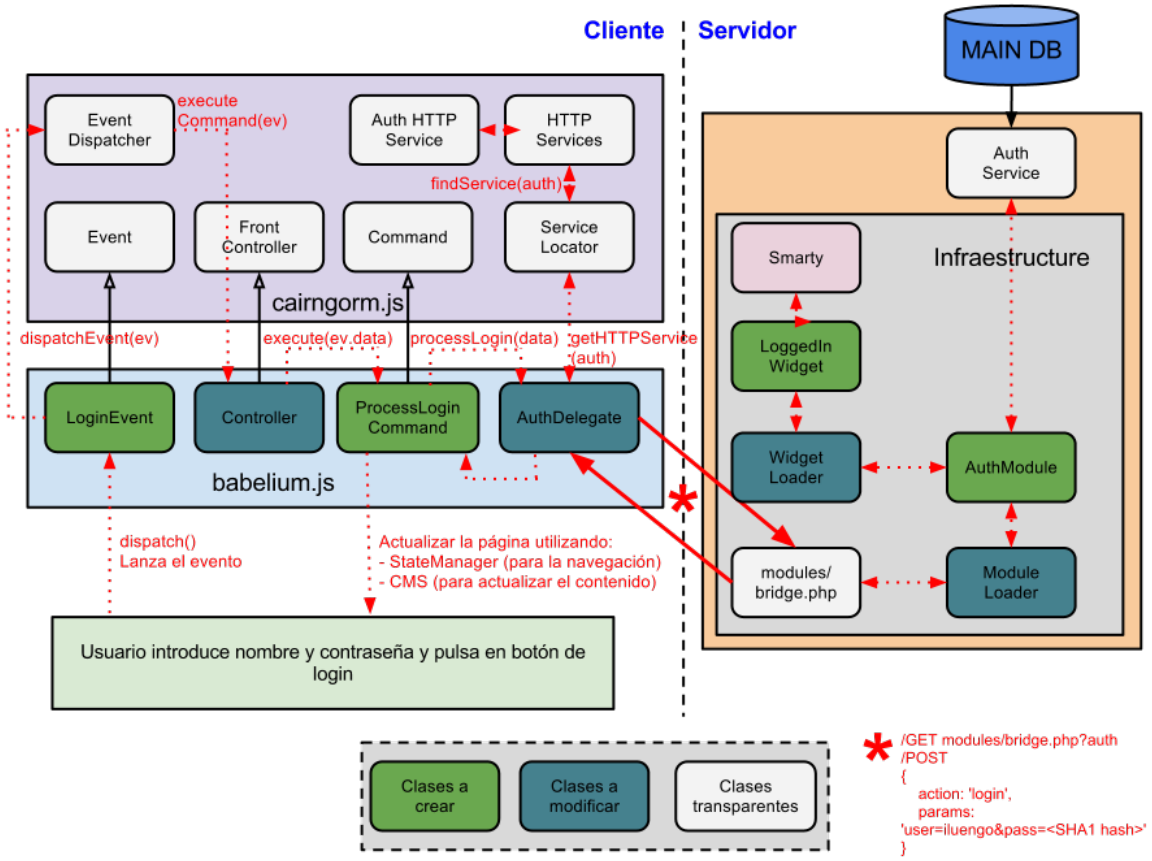


Figura 8.15: Diagrama de procesado del Login

- Las clases en blanco son las utilizadas de forma transparente, el programador ni siquiera tiene que invocarlas, forman parte de la infraestructura (ya sea Cairngorm en el cliente o php en el servidor) e interactuan para ofrecer un patrón MVC + Delegate

Además, para una mejora de diseño y usabilidad, se transformo la apariencia de la ventana de login durante la migración. En Flex era una ventana popup que aparecía tras pulsar en login, como se ve en la figura 8.17 mientras que en HTML5 se convirtió en una barra que se muestra y oculta pulsando el botón de login, más cómodo y ocupando menos espacio:

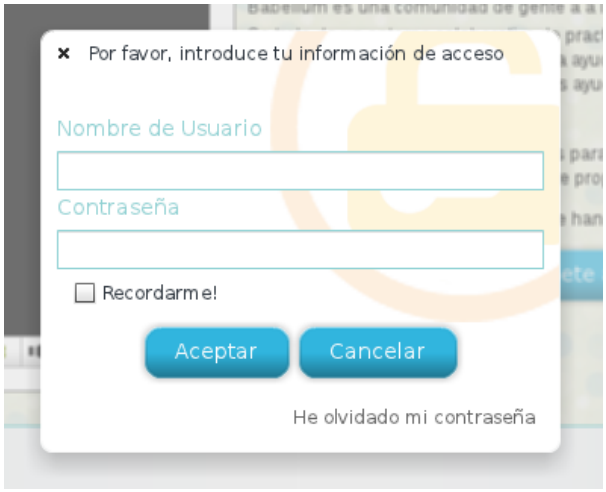


Figura 8.16: Ventana de login en Flex

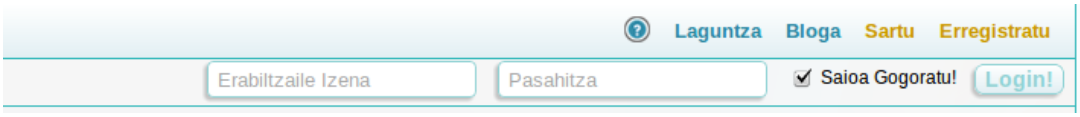


Figura 8.17: Ventana de login en HTML5

8.4.2. Registro

El proceso de registro es idéntico al de login y logout, lo único que cambia es el formulario, y los eventos/comandos y módulos/widgets que se han tenido que crear para realizarlo. Este módulo de registro fue creado por Inko Perurena, para hacer pruebas con la infraestructura de HTML5, siguiendo el ejemplo del formulario de login.

8.4.3. Validación de campos

Una vez más, HTML5 incorpora funcionalidades para facilitar tareas genéricas a los programadores. Antes había que utilizar gran cantidad de expresiones regulares para validar los campos en el cliente. Como es una tarea que se reutiliza en cada web, en HTML5 pasó a ser una implementación estandar, e implementa distintos tipos de campos con validación automática.

Los tipos de campos que soporta HTML5 son los siguientes: `button`, `checkbox`, `color`, `date`, `datetime`, `datetime-local`, `email`, `file`, `hidden`, `image`, `month`,

`number`, `password`, `radio`, `range`, `reset`, `search`, `submit`, `tel`, `text`, `time` , `url` y `week`.

Dichos campos se asignan al parámetro `type` de un componente `input` y con sólo añadir el parámetro `required` al componente, HTML5 (el navegador que lo implementa) obliga a que esté rellenado y valida su contenido (en función del tipo de campo que sea) antes de enviarlo, en caso de no ser válido o no estar rellenado, el formulario no se envía y se muestra una alerta. Ejemplo que obliga al usuario a introducir un email:

```
<input name='emailLogin' type='mail' placeholder='Inserta un email' required />
```

Si por alguna razón quisieramos forzar que la validación automática se realizase con una expresión regular propia, se puede realizar mediante el parámetro `pattern`:

```
<input type='text' name='countryCode' pattern='[A-Za-z]{3}'
    placeholder='Inserta un codigo de pais' required />
```

8.5. Home Module

El primer módulo de cara al usuario en ser migrado completamente fue la página *home*. El módulo home está preparado para recibir tanto a usuarios anónimos como a usuarios identificados. Para los primeros, se mostrarían los MOTD (*messages of the day*) a modo de ayuda y explicación de lo que se puede lograr con Babelium Project, mientras que para los segundos, se mostraría un mensaje de bienvenida acompañado de listas de los ejericios más valorados o los últimos subidos, así como la posibilidad de observar el historial de actividad.

Las figuras 8.18 y 8.19 reflejan el módulo home con el usuario identificado y sin identificar en la versión de Flex.

Figura 8.18: Home module en la versión de Flex (anónimo)

Una de las diferencias más llamativas de la versión HTML5 frente a la versión de Flex, es que mientras en esta última la web ocupaba el 100 % de ancho y alto de la web, la versión HTML5 tenía un ancho máximo, para mantener la web con un diseño elegante aún en resoluciones muy grandes, donde estirada al 100 % quedaría descolocada.

En las siguientes subsecciones se mostrará el resultado de la migración del módulo, tanto para los usuarios identificados como sin identificar, mostrando también los componentes que han tenido que ser migrados de la versión de Flex.

Figura 8.19: Home module en la versión de Flex (identificado)

8.5.1. Infraestructura

En la figura 8.20 se muestran los elementos (del mismo nombre) que se han tenido que migrar de Flex a HTML5 para habilitar el módulo home. En el diagrama, se obvian las plantillas HTML5 y CSS3 que cargan cada uno de los widgets, pues es Smarty el que los gestiona; y la infraesturua de la aplicación, tanto en el cliente como en el servidor, explicada ya en el punto 8.2.

- Se ha creado un módulo Home en el servidor, que tiene dos estados principales posibles: identificado y anónimo; y 3 estados secundarios en caso de que el usuario esté identificado: mejores vídeos, últimos vídeos (por defecto) y última actividad.
- Se han creado widgets para cada uno de los estados principales, encargados de mostrar los MOTD en caso de estar identificado, o no.
- En caso de estar identificado, además de los MOTD, se carga un widget dependiendo del estado secundario en el que esté el módulo; si no se especifica ninguno, se cargan los últimos vídeos por defecto
- Para gestionar los cambios de estado se ha creado el evento `HomeEvent` y los comandos correspondientes a cada subsección: `Best Videos Command`, `Latest`

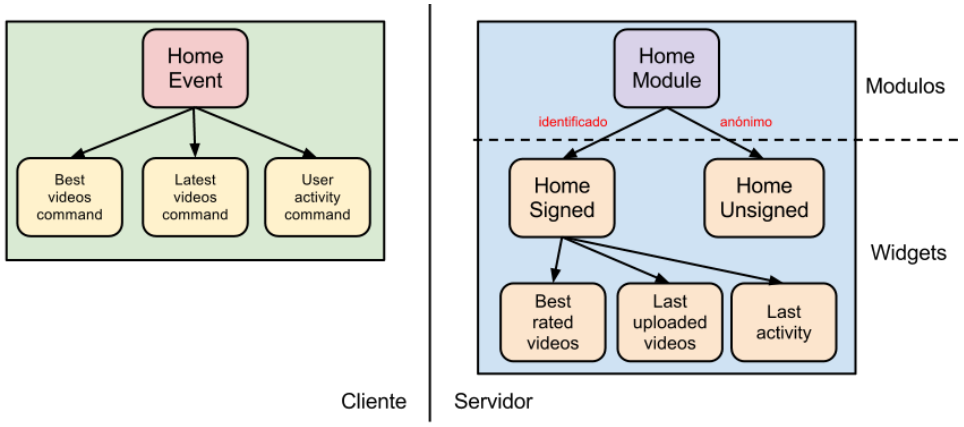


Figura 8.20: Estructura del módulo home

`Uploaded Videos Command` y `Latest User Activity Command`; que estando en el cliente, lanzarán una cadena de eventos de Cairngorm para cargar dinámicamente el contenido del módulo.

Explicada la infraestructura, las siguientes subsecciones comentarán las peculiaridades de la migración y el resultado de la misma.

8.5.2. Home sin identificar

Siguiendo el mismo modelo que la versión de Flex, el diseño contempla los mensajes del día a modo de ayuda para los nuevos usuarios. Éstos mensajes que se pueden cambiar pulsando en los botones inmediatamente superiores a ellos, se crearon utilizando el componente `Tabs` de JQuery. Permite, teniendo X botones, crear un contenedor para cambiar su contenido a X posibles. Los botones hacen las veces de pestañas en lo que sería un panel común de estas características, sólo que con un diseño modificado para una apariencia más simple. En la figura 8.21 puede verse el resutlado de la migración del módulo.

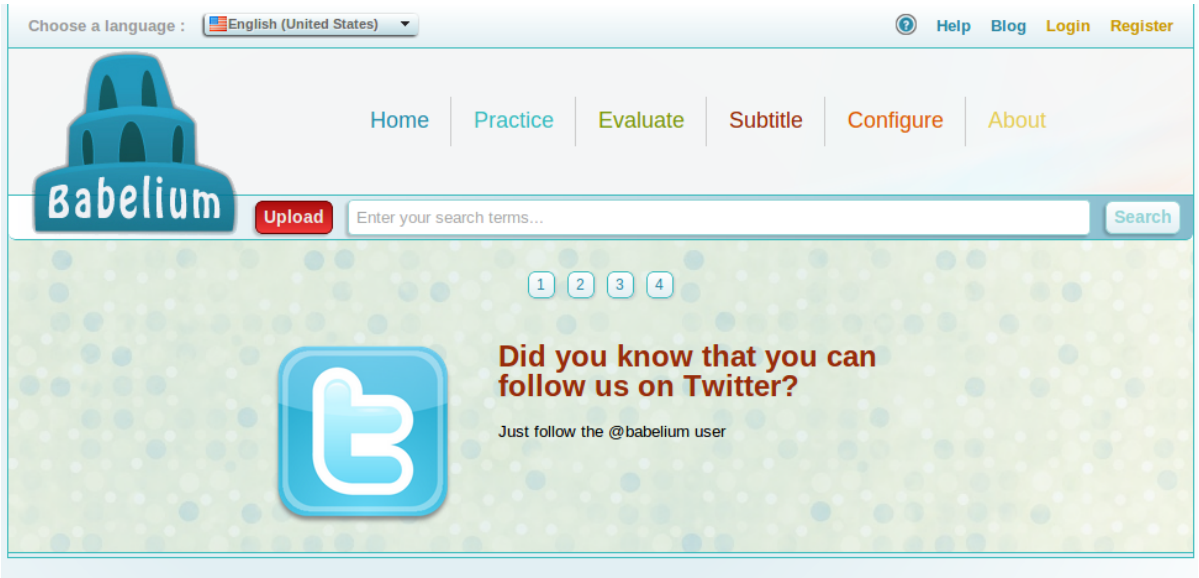


Figura 8.21: Home de la versión HTML5 (anónimo)

8.5.3. Home identificado

El módulo home cuando el usuario está identificado, muestra un mensaje de bienvenida junto a una lista de acciones posibles dentro de Babelium Project. Esta parte de la web se ha migrado sin complicaciones, pues es una simple plantilla HTML5+CSS3. Sin embargo, el módulo home para un usuario identificado carga, además, 3 pestañas con 3 contenidos posibles.

Pese a lo que pueda parecer, las pestañas mencionadas y sus contenidos, no se han implementado con un `TabPanel`, pues el contenido que estas pestañas es demasiado grande para tenerlo en memoria en el cliente, así que se decidió implementarlo como enlaces corrientes, que cambiarían el contenido del módulo dinámicamente mediante Cairngorm.

En la figura 8.22 se puede observar el resultado del módulo una vez el usuario se ha identificado.

Figura 8.22: Home de la versión HTML5 (identificado)

Best videos y Last uploaded videos

Para crear la lista de los mejores vídeos y últimos vídeos subidos (así como la lista de vídeos del módulo *practice*, que se explirará más adelante), se utilizó una plantilla HTML5+CSS3 genérica, mediante Smarty, que nos ahorró tener que reescribir código, ya que permite reutilizarla tanto como queramos mediante inserción de variables y

declaración de ellas en tiempo de *parseo*[2].

Como resultado de reutilizar la misma plantilla para mostrar la información de vídeos, podemos observar en la figura 8.23 la lista de los últimos vídeos subidos.

Figura 8.23: Home HTML5: últimos vídeos subidos

Latest user activity

Para mostrar la última actividad de un usuario, se hizo uso de las DataTables (sección 7.3.3), dando como resultado una tabla ordenable, paginada y filtrable con la última actividad del usuario, como puede observarse en la figura 8.24.

[2]Analizar una entrada de texto para transformarla en otra distinta o en una estructura de datos que pueda ser procesada.

Figura 8.24: Home HTML5: última actividad del usuario

8.6. Practice Module

El módulo *practice* de Flex está pensado para grabar ejercicios de doblaje, es decir, un usuario puede acceder a una lista de vídeos (ejercicios). Entre ellos, puede elegir el deaseado y entrar al modo preview, donde podrá ver el vídeo con los subtítulos en su correspondiente idioma. Acto seguido, el usuario podrá decidir grabar un ejercicio doblando la voz y, opcionalmente, grabando su webcam.

A continuación se muestran 3 figuras, 8.25, 8.26 y 8.27 con la correspondiente lista de vídeos, preview y grabación de ejercicios.

Figura 8.25: Módulo practice en la versión de Flex

Una vez el usuario ha grabado el ejercicio, puede volver a verlo para ver como ha quedado, repetir la grabación, descartarlo o guardarlo. En el caso de salvar la grabación para que se la evalúen, se le restará una cantidad de créditos a su cuenta a modo de coste.

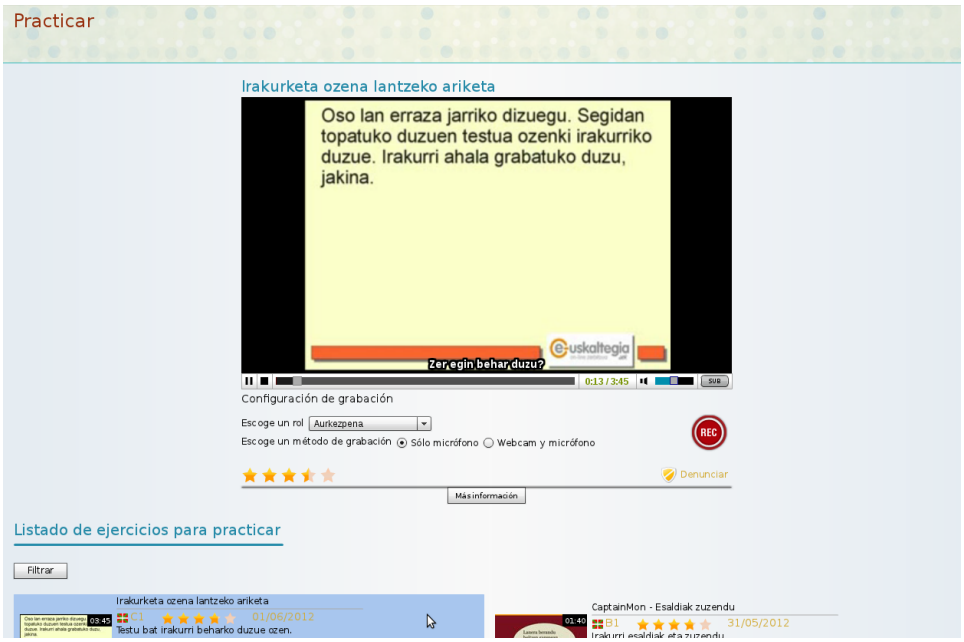


Figura 8.26: Módulo practice (Flex): preview de un ejercicio

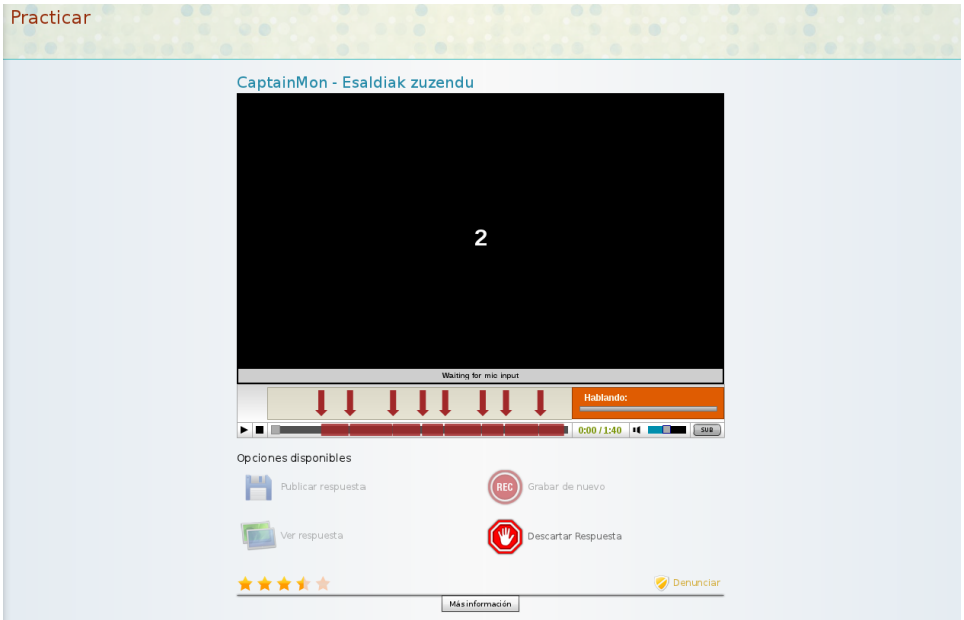


Figura 8.27: Módulo practice (Flex): grabación de un ejercicio

8.6.1. Infraestructura

Se puede observar la infraestuctura, compuesta por una serie de eventos y comandos en el cliente, y modulos y widgets en el servidor; en la figura 8.28. En el diagrama, se obvian las plantillas HTML5 y CSS3 que cargan cada uno de los widgets, pues es Smarty el que los gestiona; y la infraesturua de la aplicación, tanto en el cliente como en el servidor, explicada ya en el punto 8.2.

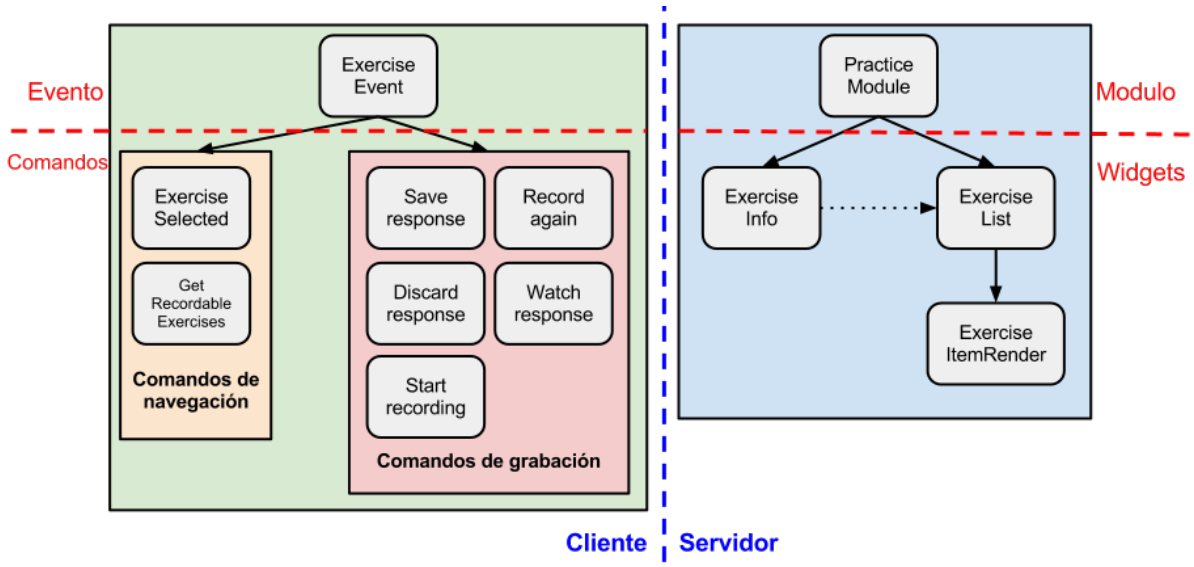


Figura 8.28: Estructura del módulo practice

El módulo practice está compuesto principalmente por 2 pantallas, o estados: lista de vídeos (o ejercicios) y preview (o información) del ejercicio. Dentro del preview del ejercicio, el usuario puede elegir comenzar a grabar el ejercicio, y entraría en modo de grabación, pero esto no se contempla como cambio de estado en el módulo, ya que no requiere de ninguna acción en el servidor, tan solo interacción con el reproductor para comenzar a grabar.

- Se ha creado un módulo `Practice Module`, con dos estados principales: ver la lista de vídeos y ver el preview del vídeo
- Se han creado widgets para cada uno de los estados, el primero para mostrar una lista de vídeos (el cual utiliza el *ItemRender* mencionado en la sección 8.5.3), y el segundo para mostrar información y opciones de grabación de un ejercicio seleccionado

- Para gestionar la selección de ejercicios y la muestra de vídeos, se han creado dos comandos: `Get Recordable Exercises` y `Exercise Selected`; que, respectivamente, obtienen los ejercicios en el idioma que el usuario quiere aprender, y seleccionan un ejercicio para su preview.

- Además, para gestionar la grabación del ejercicio una vez el usuario lo ha escogido, se han creado 5 comandos que controlan: iniciar, descartar, guardar, repetir grabación y ver la grabación antes de enviarla.

8.6.2. Lista de ejercicios

La lista de ejercicios, consiste en una lista de todos los ejercicios posibles (para el idioma en el que el usuario desea practicar, o todos si el usuario no está identificado) ordenados por fecha de subida. Mediante el plugin para JQuery JPList (sección 7.3.2) y la plantilla genérica para el renderizado de ejercicios (sección 8.5.3), se creó una lista paginable (con 10 ejercicios por página) y filtrable mediante título o descripción. La figura 8.29 muestra el resultado.

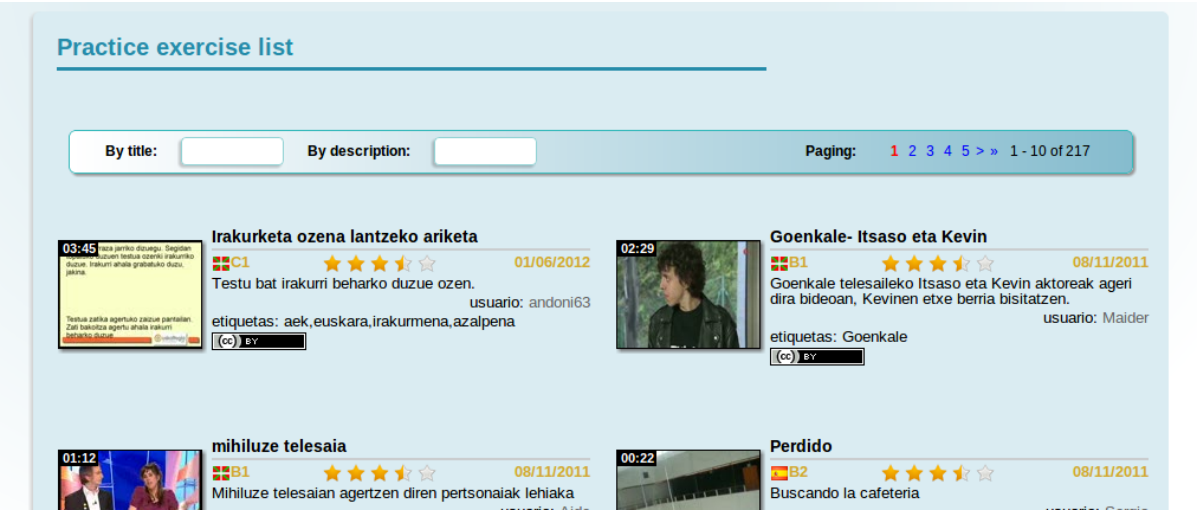


Figura 8.29: Módulo practice (HTML5): lista de ejercicios

Los vídeos se resaltan cuando el usuario pasa el ratón por encima, y una vez el usuario ha escogido un ejercicio que le atraiga, puede pinchar en él para entrar a ver el preview.

8.6.3. Preview de ejercicios

En el estado de preview, el usuario puede ver el vídeo completo junto a los subtítulos en los idiomas disponibles. Además, al lado del vídeo aparecerán una lista de etiquetas del vídeo y las opciones de grabación, entre las que se encuentran:

- Idioma del vídeo
- Rol escogido para el doblaje del ejercicio (de los que participan en el vídeo)
- Grabar sólo con micrófono, o también con webcam

Para realizar dicha sección, se ha utilizado únicamente layout y estilos HTML5+CSS3 y el reproductor de vídeo, mencionado en la seccion 7.1, además de JRaty (sección 7.3.4) para la inserción del widgets de valoración del ejercicio. En la figura 8.30 se muestra el resultado de la previsualización.

Figura 8.30: Módulo practice (HTML5): preview de un ejercicio

Una vez el usuario ha visualizado un ejercicio acorde a su nivel y de su gusto, puede elegir las opciones deseadas, y comenzar a grabar un ejercicio.

8.6.4. Reproducción y grabación de ejercicios

La reproducción y grabación de vídeos se hace mediante el reproductor de vídeo mencionado en la sección 7.1. Dado que ofrece una API JavaScript, se creó una clase llamada `ExcerciseManager` (o *BP.EM*) encargada de gestionar las acciones del reproductor, y que es utilizada por los comandos de grabación mencionados al inicio de la sección.

`ExerciseManager` hace de pasarela entre JavaScript y la API externa del reproductor. Ofrece métodos como `load exercise` o `start recording` que abstraen de la API y el uso del reproductor, permitiendo fácilmente previsualizar o grabar un ejercicio, con un sólo método.

Una lista resumida de los métodos más útiles es la siguiente:

loadSelectedExercise. Cuando un usuario selecciona un ejercicio, lo carga e inicia en el reproductor su previsualización

showArrows/hideArrows. Muestra u oculta las flechas del reproductor, donde indica cuando habla el rol seleccionado.

setupRecording. Prepara el reproductor y entra en modo de grabación.

showRecordingOptions. Muestra las opciones posibles a realizar durante y tras la grabación.

watchResponse. Visualiza la grabación que el usuario acaba de realizar.

saveResponse. Guarda la grabación.

Aunque hay más métodos secundarios para que estos funcionen (como la obtención de subtítulos o roles de la base de datos), estos son los principales, que desencadenan desde el inicio hasta el final de la grabación. En la figura 8.31 se puede observar una grabación de un ejercicio:

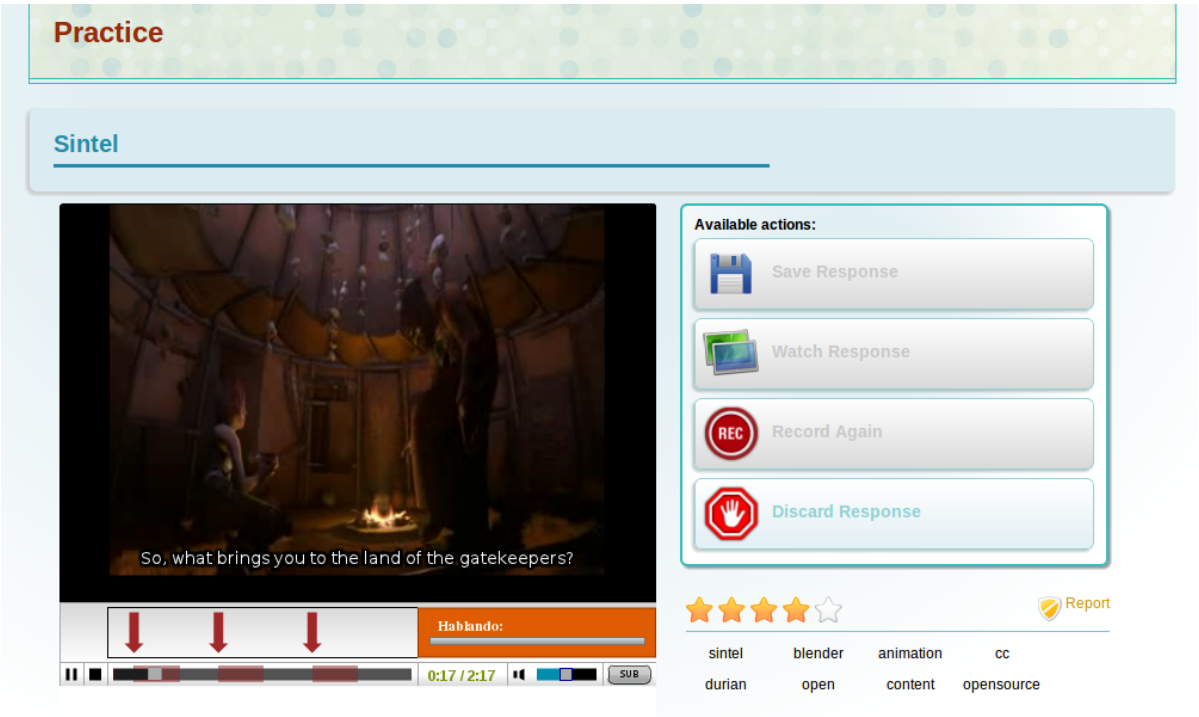


Figura 8.31: Módulo practice (HTML5): grabación de un ejercicio

8.7. Localization

Con fuente en la wikipedia, **localization** o **localisation**:

> *Language localisation: the process of translating a product into different languages or adapting a product for a specific country or region*

Babelium Project, al ser una plataforma con la finalidad de aprender idiomas, tiene que poder ofrecer sus servicios varios idiomas para poder adaptarse a todos los usuarios. Es por ello que, al igual que en la versión en Flex, se creó para HTML5 un sistema de localization, para poder traducir dinámicamente la web a varios idiomas, sin tener que reescribir la web en todos y cada uno de los idiomas.

8.7.1. Antecedentes

El funcionamiento del módulo localization en Flex era el siguiente:

La aplicación contaba con ficheros de texto, uno por cada idioma, donde todas las frases que aparecían en la aplicación, estaban identificadas por un nombre y un valor (de la misma forma que funciona una tabla hash: key - value). Por ejemplo, se muestra a continuación unas pocas líneas del fichero en inglés (en_US) que hacen referencia a los nombres de las secciones de la página:

```
1 LABEL_HOME=Home
2 LABEL_EXERCISES=Practice
3 LABEL_EVALUATIONS=Evaluate
4 LABEL_SUBTITLE=Subtitle
5 LABEL_CONFIGURATION=Configure
6 LABEL_SEARCH=Search
7 LABEL_UPLOAD=Upload
8 LABEL_ABOUT=About
```

Y su traducción equivalente en el fichero en castellano (es_ES):

```
1 LABEL_HOME=Inicio
2 LABEL_EXERCISES=Practicar
3 LABEL_EVALUATIONS=Evaluar
4 LABEL_SUBTITLE=Subtitular
5 LABEL_CONFIGURATION=Configuraci n
```

```
6 LABEL_SEARCH=Buscar
7 LABEL_UPLOAD=Subir
8 LABEL_ABOUT=Acerca de Babelium
```

Dentro del código ActionScript, Flex dispone de un `resourceManager` que controla el idioma en el que está la página actualmente, y, por ejemplo, para el módulo home, en lugar de poner en el título de la sección "Home" o "Inicio" manualmente, se utiliza el `resourceManager` que se encargará de introducir el titulo correspondiente al idioma en el que se encuentra, cogiendo los datos de un fichero u otro:

```
1 resourceManager.getString('LABEL_HOME')
```

Si la página está en inglés, `resourceManager` mostrará Home cómo titulo de página, si está en castellano mostrará Inicio, y si por el contrario está en otro idioma, buscará en su fichero de traducción correspondiente y lo sustituirá para mostrarlo en pantalla.

De la misma forma que en Flex, se implementó un sistema de localización para la versión HTML5. Sin embargo, a diferencia de la versión en Flex, que tenía todo el contenido en el cliente, la versión HTML5 disponía de la parte de generación de contenidos (en el servidor) y la parte de actualización de contenidos (en el cliente), así que se crearon dos sistemas (con el mismo funcionamiento) de localización; uno para generar contenidos en distintos idiomas (en el servidor) y otro para mostrar mensajes de carga y errores (en el cliente).

8.7.2. Localization en el servidor

Para no cambiar el sistema de localización de Babelium de Flex a HTML5, y tener que traducir la web de nuevo, se decidió reutilizar los ficheros de traducción (explicados en el apartado anterior) y desarrollar un sistema para utilizarlos. Dicho esto, los ficheros de traducción siguieron intactos y se reutilizaron exactamente con la misma funcionalidad en la versión HTML5.

Para que esto pudiera ser posible fueron necesarios dos pasos:

1. Gestión del idioma de la página: que se realizó mediante el *language bar* (figura **??**) y las sesiones de PHP, donde al cambiar el idioma de la web mediange el *language bar*, se actualizaba en la sesión del usuario una variable que contenía el idioma en el que se debía mostrar la página.

2. Muestra de contenido en el idioma de la página: dado que la gestión de contenidos se realizaba mediante Smarty, se decidió realizar un plugin para Smarty (i18n) capaz de traducir variables utilizando los ficheros de traducción importados de la versión Flex.

Plugin i18n

Este plugin funciona mediante Smarty, y su funcionamiento es bastante simple a la vez que cómodo. Disponemos de plantillas Smarty (.tpl escritos en HTML5+CSS3 con funciones y variables Smarty), disponemos ficheros de traducción para cada idioma soportado de Babelium Project y disponemos de una variable en la sesión del usuario indicando en que idioma debe ser visualizada la Web.

Dicho esto, el funcionamiento del plugin consiste en incluir un comando Smarty (creado especialmente para Babelium Project) e indicándole el nombre del texto que se quiere insertar. Acto seguido, el plugin buscará en la sesión del usuario el idioma en el que visualizar la web, buscará en dicho fichero la variable y la sustituirá por su valor. La sintaxys del plugin es la siguiente:

```
1 {i18n name="<nombre de variable>"}
```

Si queremos insertar el nombre del módulo de evaluación, y que se traduzca dinámicamente al idioma en el que el usuario ve la página web, bastaría con insertar dentro de una plantilla HTML la función arriba mencionada:

```
1 <span>{i18n name="LABEL_EVALUATION"}</span>
```

El plugin reemplazaría esa función por el valor de LABEL_EVALUATION en el idioma del usuario.

8.7.3. Localization en el cliente

El funcionamiento es prácticamente el mismo, existen ficheros para cada idioma con mensajes de carga y error de cada acción/sección de la página. Sin embargo, al estar el cliente cargado en JavaScript, el plugin creado para Smarty no funciona esta vez. Dicho de otra forma, lo que el cliente recive, es una librería JavaScript. Para poder aplicarle localización, se realizaron las siguientes acciones:

1. Siguiendo la sintaxis anterior, insertar dentro de JavaScript contenido que debe ser traducido a modo de variable, con la siguiente sintaxis:

```
alert("{{$ERROR_LOADING_HOME_MODULE}}");
```

2. Crear un script PHP llamado `resources.php` que sirviera el script JavaScript en el lenguaje del usuario

Dicho de otra forma, antes se incrustaba el script JavaScript directamente en la página (*babelium.js*):

```
<script type="text/javascript" src="babelium.js"></script>
```

Mientras que ahora, hay un paso intermedio. Lo que se inserta es el programa PHP `resources.php` con el parámetro *babeliumjs*, que lo que hace es *parsear* por completo el script de *babelium.js*, y aplicándole transformaciones de Smarty + localization (de la misma manera que en el apartado anterior), devolver al cliente el script *babelium.js* traducido a idioma en el que está viendo la página:

```
<script type="text/javascript" src="resources.php?babeliumjs"></script>
```

8.8. Evaluation Module

El módulo de evaluación, creado para evaluar y ver evaluaciones de los ejercicios grabados por otros usuarios, o los tuyos mismos. Es requisito indispensable que el usuario esté identificado para acceder al módulo de evaluación, en caso contrario se cargará el Widget *forbidden* diciéndole que necesita identificarse para poder navegar por el módulo.

Una vez identificado el usuario, podrá navegar por el módulo, en el cual observará 3 pestañas: pendientes de evaluación, respuestas que me han evaluado y respuestas que he evaluado. En cada una de las pestañas, siendo la primera la pestaña por defecto, se cargará una lista de ejercicios, ya sea para evaluar (pendientes e evaluación) o para ver/revisar (evaluados por mi o evaluados a mi).

En la figura 8.32 se observa el estado inicial del módulo de evaluación, en la figura 8.33 se observa el menu para evaluación de un ejercicio, y en la figura 8.34 se puede observar la revisión de un ejercicio evaluado por nosotros.

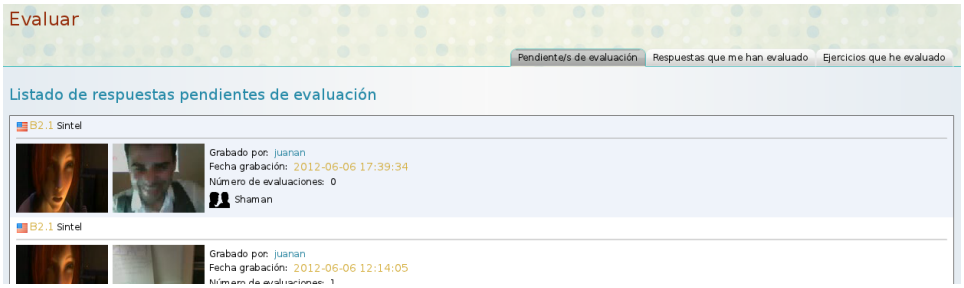


Figura 8.32: Módulo evaluación en la versión de Flex

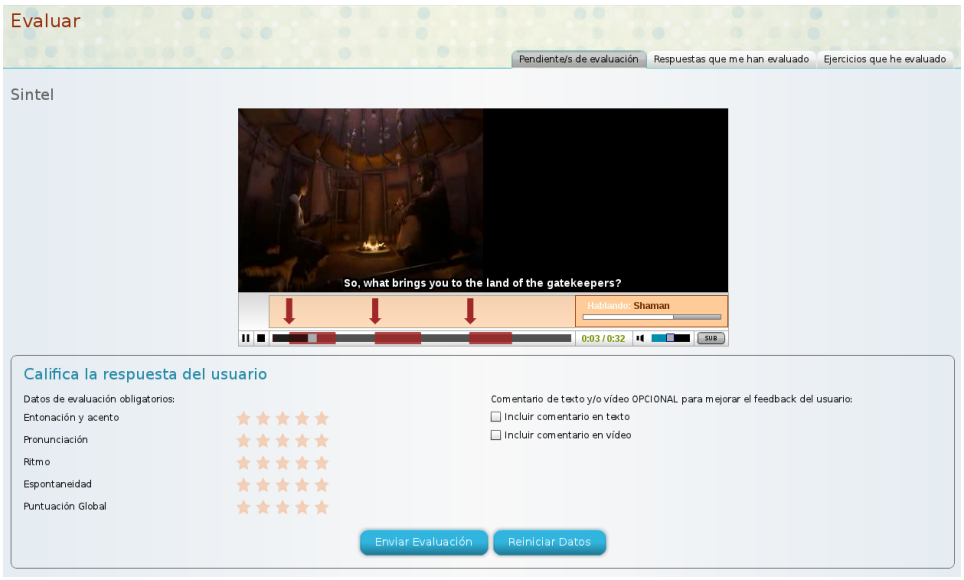


Figura 8.33: Módulo evaluación (Flex): evaluando de un ejercicio

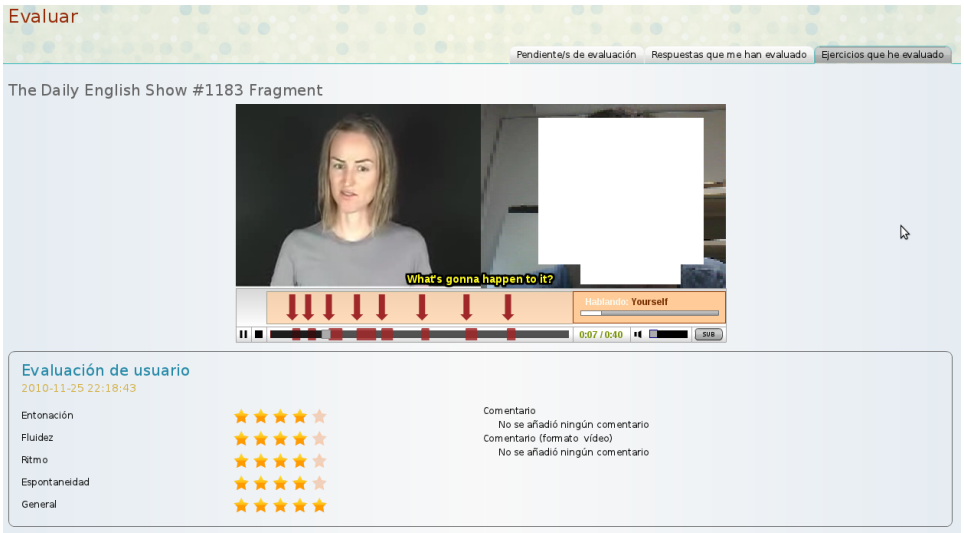


Figura 8.34: Módulo evaluación (Flex): revisando la evaluación de un ejercicio

8.8.1. Infraestructura

El módulo tiene una lista de pestañas y una lista de evaluaciones que se mostrarán siempre, variando sólo el contenido de esta última lista, dependiendo del estado en el que se encuentre el módulo. Las pestañas controlan lo que el usuario desea realizar: ver ejercicios que le han evaluado, ver ejercicios que él ha evaluado, o ver ejercicios pendientes de evaluación.

La infraestructura en el servidor la forman un módulo, Evaluation Module, y distintos widgets: un menú con pestañas para selección de acción a realizar en el módulo y una lista genérica que mostrará listas de ejercicios, independientemente del estado en el que se encuentre el módulo; y luego, una vez el usuario haya elegido un ejercicio, se mostrará uno de los 3 widgets: Waiting for assesment (para evaluar un ejercicio), Assessed to user (para ver la revisión realizada a un ejercicio suyo) y Assessed by user (para revisar una evaluación realizada por él).

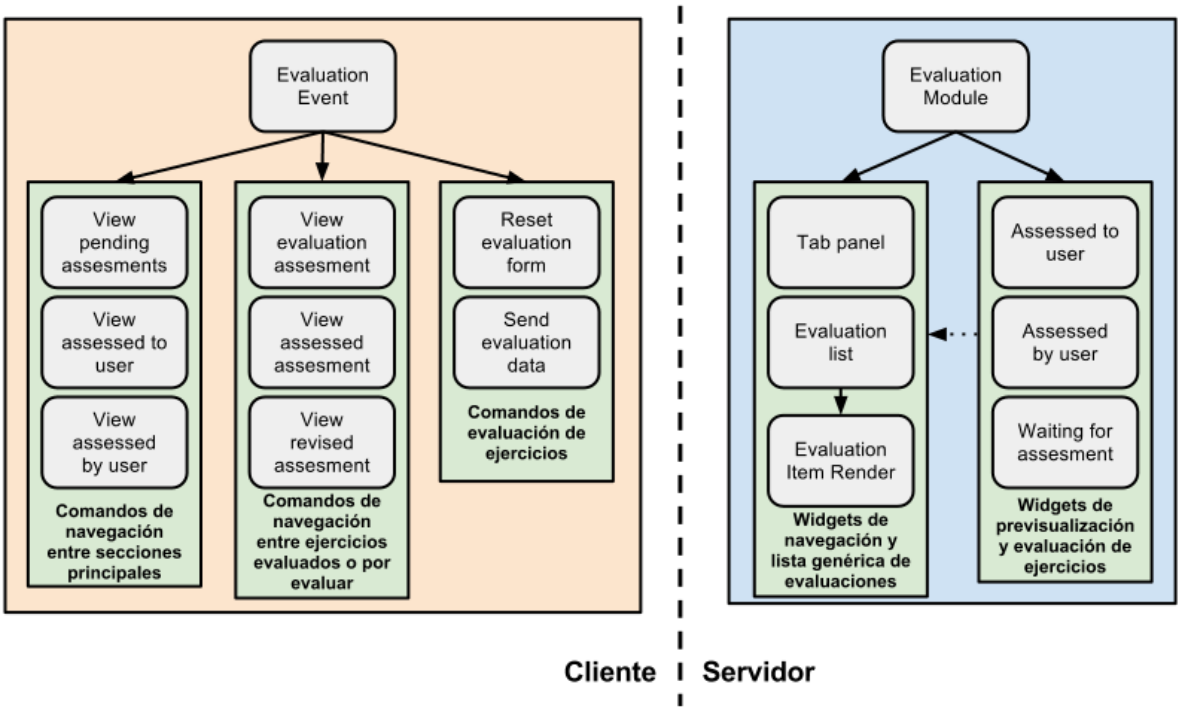


Figura 8.35: Estructura del módulo evaluation

En el cliente, siguiendo la estructura Cairngorm, existe un evento que gestiona todas las acciones `EvaluationEvent`, y distintos comandos con 3 tipos de funcionalidades:

1. Navegar entre las listas de ejercicios: ver los ejercicios pendientes de evaluación, ver los ejercicios que el usuario ha evaluado, o ver los ejercicios que le han evaluado al usuario

2. Ver información concreta de un ejercicio: para evaluarlo, para revisar una evaluación hecha o para revisar una evaluación que le han hecho al usuario.

3. En el caso de encontrarse evaluando un ejercicio, hay otros dos comandos para gestionar el descarte y el guardado de la evaluación.

La figura 8.35 refleja lo explicado, obviando elementos de la infraestructura, tanto en el cliente como en el servidor.

8.8.2. Lista de evaluaciones

Las listas de ejercicios pendientes de evaluación, ejercicios evaluados por el usuario o ejercicios evaluados al usuario, están formadas por una única lista genérica, con un mismo *ItemRender* que cambia el contenido de su lista de evaluaciones en función de uno de los 3 estados mencionados, controlados por las pestañas del módulo.

Al pulsar sobre una de las pestañas, ejercicios evaluados por el usuario por ejemplo, se disparará `EvaluationEvent` y lanza una serie de acciones mediante Cairngorm que termina en la actualización de la lista insertando en ella las evaluaciones correspondientes.

La lista de evaluaciones es un widget creado utilizando JPList, explicado en la sección 7.3.2, y muestra ordenados por fecha de realización los ejercicios, filtrables por nombre o descripción y paginados mostrando 10 ejercicios por página. Se puede observar el resultado en la figura 8.36, utilizando para el ejemplo la lista de ejercicios pendientes de evaluación.

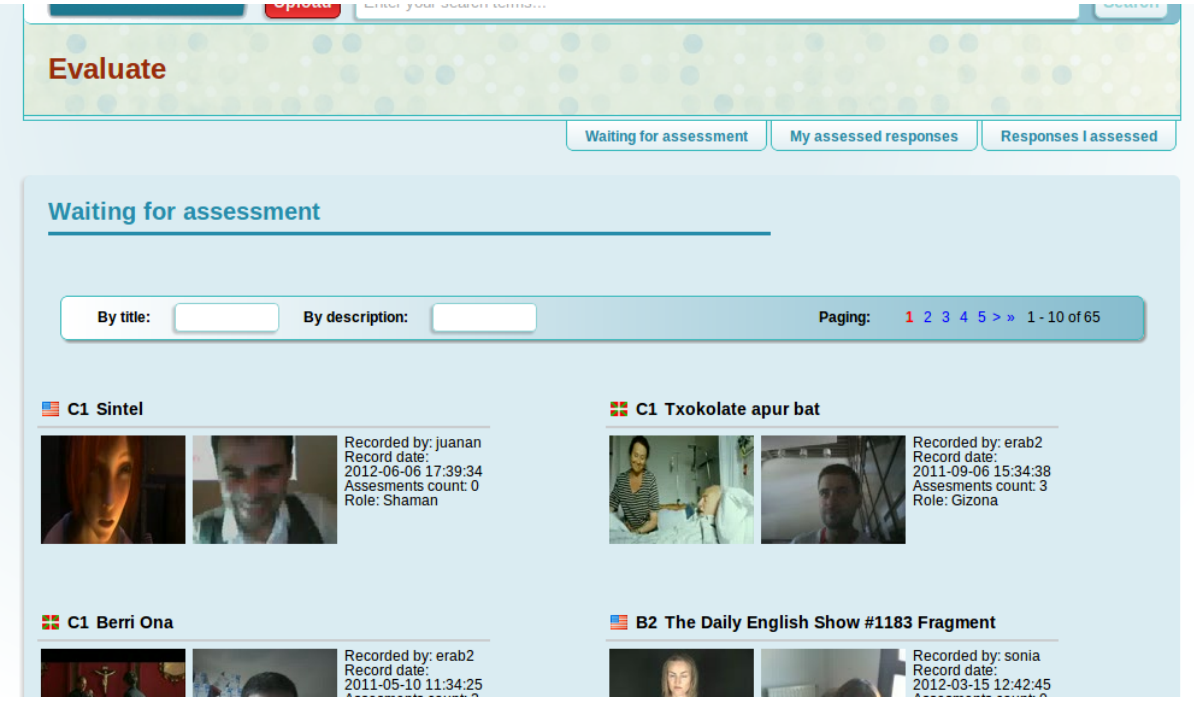


Figura 8.36: Módulo evaluate (HTML5): lista de ejercicios

8.8.3. Evaluación de ejercicios

Si el usuario escoge un ejercicio de la lista de ejercicios pendientes de evaluación y clicka en él, entrará a la pantalla de evaluación, donde podrá evaluar el ejercicio mediante puntuaciones de 1 a 5 en distintos matices (pronunciación, entonación...). Adicionalmente, en caso de asi desearlo, podrá añadir un comentario para felicitar al usuario o ayudarle a mejorar.

Para realizar este widget, se ha utilizado layout y estilos HTML5+CSS3 y el reproductor de vídeo, mencionado en la sección 7.1, además de JRaty (sección 7.3.4) para la evaluación del ejercicio mediante valoraciones en forma de estrellas. En la figura 8.37 se muestra el resultado.

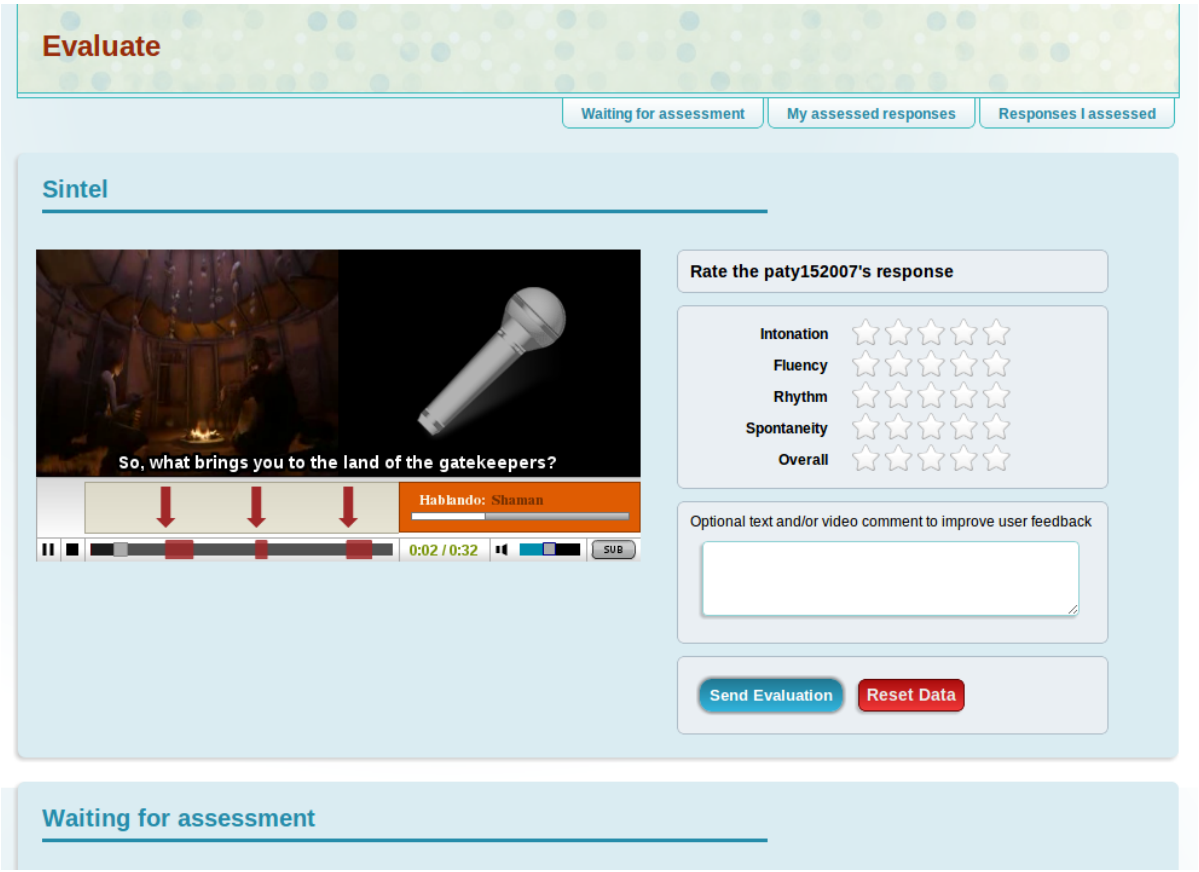


Figura 8.37: Módulo evaluate (HTML5): evaluación de ejercicios

8.8.4. Visualización de evaluaciones

Por último, el usuario puede navegar mediante las pestañas a la lista de ejercicios que le han evaluado o que él ha evaluado, y ver los detalles de dicha evaluación. El widget es muy parecido al anterior, con la diferencia de que aquí el usuario, no podrá alterar la valoración; es decir, la puntuación de las estrellas es estática y únicamente *read-only*.

El widget está formado por el reproductor de vídeo (seccion 7.1), JRaty (sección 7.3.4) y una plantilla HTML5+CSS3 para estilizar la forma en la que se presentan los resultados:

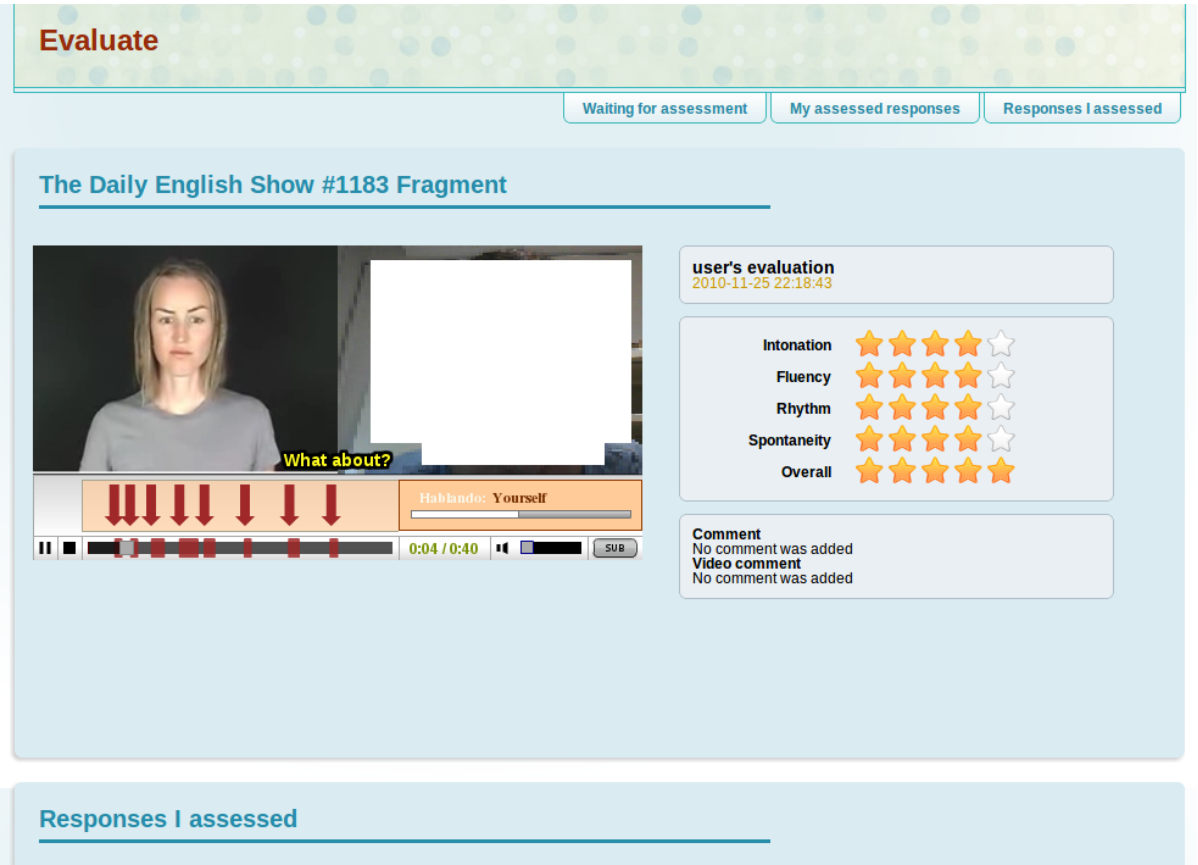


Figura 8.38: Módulo evaluate (HTML5): revisión de ejercicios

Capítulo 9

Conclusiones

Al finalizar el proyecto, se han logrado todos y cada uno de los objetivos que se propusieron al comienzo del mismo, teniendo como resultado una web en HTML5 completamente operativa, con todas las funcionalidades principales de Babelium Project que se propusieron migrar.

Tras el proyecto, he obtenido grandes conocimientos en el campo de aplicaciones web, tanto realizando y entendiendo como funcionan las infraestructuras en la parte del servidor, y creando una adecuada para el caso que nos ocupa; así como programando patrones MVC+Delegate para el cliente. He usado cantidad de lenguajes de programación y frameworks distintos, para que todo el proyecto fuese totalmente operativo, y ha sido una muy grata experiencia ver como todos ellos encajaban perfectamente. Ha sido la primera vez que me enfrentaba al diseño completo de la arquitectura de una aplicación de estas características, sin embargo, todo ha acabado saliendo bien y la aplicación es completamente funcional y soportada por todos los navegadores.

En cuanto la temática del proyecto, el conjunto de tecnologías que forman HTML5, queda decir que aún está en un estado bastante inmaduro. Si bien los navegadores ya se han puesto las pilas para implementar los estándares de CSS3 y HTML5, no son coherentes entre ellos, y no todas las funcionalidades que funcionan en un navegador, funcionan de la misma manera en otro. Por ejemplo, hay propiedades CSS3 que funcionan en un navegador, que no funcionan en otro, o incluso, que funcionan en ambos navegadores, pero no con la misma sintaxis, lo que conlleva a malgastar tiempo y escribir código redundante para que todos los navegadores puedan ver lo mismo.

Además, HTML5 es mucho más que un conjunto de etiquetas y estilos, el uso de los campos `-data`, el uso de APIs como `file` o `history`... todo ello está siendo implementado por los navegadores, sin embargo, los programadores web seguimos teniendo el mismo problema que desde hace una década: no podemos controlar los navegadores que utilizan los usuarios. Aún hay usuarios utilizando navegadores web prehistóricos, lo que obliga a todo desarrollador web, a establecer *fallbacks* para éstos.

Por último, otra parte importante del proyecto ha consistido en la implementación de un método para reproducir y grabar audio y vídeo vía web, sin embargo, y una vez más debido a la inmadurez de HTML5, no se pudo implementar en HTML5 y tuve que utilizar un widget en Flash para ello.

Aún así, y pese a todas las complicaciones, como resultado del proyecto se puede afirmar que es posible y completamente factible migrar una aplicación a los nuevos estándares. De forma más o menos complicada, pero es posible. En este proyecto se ha conseguido realizar aplicando *fallbacks* para los estilos CSS3 y los navegadores obsoletos (ofreciéndoles una alternativa menos vistosa en CSS2) e implementando la grabación de audio y vídeo en Flex (sólo el reproductor) y comunicarlo mediante JavaScript con el resto de la página web (gracias al `ExternalInterface`).

Capítulo 10

Trabajo Futuro

Contents

El trabajo futuro dentro del proyecto en HTML5 podría tener distintas nuevas líneas de desarrollo, como podrían ser las que se muestran a continuación, de cara al futuro:

10.1. Adaptación del reproductor a HTML5

A lo largo de la memoria se ha mencionado que HTML5 no está totalmente desarrollado, y que en estos momentos no está soportada la grabación ni de audio ni de vídeo. Sin embargo, HTML5 está en constante desarrollo, y puede que pronto alcance un grado de madurez suficiente que nos haga cambiar de opinión, y empezar a plantearnos seriamente el migrar el reproductor y grabador de vídeo a este conjunto de tecnologías.

Una vez llegue este punto, la web pasaría a estar formada completamente por HTML5 (a excepción de para aquellos navegadores que aún no la soporten).

10.2. Implementación de nuevas funcionalidades

Aún quedan algunas funcionalidades de la versión de Flex sin implementar en la versión de HTML5, como el módulo subtitles o upload que no estaban contemplados en este proyecto.

Sería interesante estudiar la posibilidad de implementar el módulo de subtitulación, y con ello la posibilidad de obtener y crear subtítulos para los módulos de evaluation y practice, mediante la iniciativa de mozilla: http://www.universalsubtitles.org/es/. Parece todo un reto y nada fácil de realizar, pero podría ser interesante coordinar los subtítulos de Babelium Project con una iniciativa libre de tal magnitud.

Por otra parte, también sería interesante estudiar el estado y la viabilidad de implementar el módulo upload mediante el File API de HTML5: una API para ver/editar/-controlar los ficheros del ordenador local, y luego mediante los formularios de HTML5 enviarlos y recibirlos en PHP.

10.3. Cairngorm JS como proyecto independiente

Ahora mismo, Cairngorm JS es una implementación de apenas 7 clases para ofrecer el patrón MVC (y también podría añadirse el patrón Delegate). Sin embargo, creo que

podría abrirse la librería al público. Con un poco de depuración, un poco más de desarrollo y añadiendo trozos de código para el control de errores, podría llegar a ser una librería útil para otros proyectos, aunque estos proyectos no sean migraciones de Flex.

10.4. Documentar y abrir la API

Ahora mismo, aunque la API está completamente terminada y funcional (a modo de ejemplo está esta versión HTML5 que funciona utilizando la API desde el cliente), aún no está documentada ni abierta al público del todo. Creo que sería un buen paso documentar bien la API de Babelium Project para que pueda ser usada por otros desarrolladores, ya que podrían crearse distintos plugins para integrar Babelium Project en otras plataformas.

A modo de ejemplo está el plugin que el propio grupo de investigación GHyM ha creado para Moodle [30].

Apéndice A

Detalles técnicos del Proyecto

Contents

A.1. Código fuente

Antes de comenzar a leer los anexos, a continuación se muestran los repositorios donde reside el código del proyecto, para poder ver el respositorio y el contenido de los scripts mientras se leen los detalles técnicos de los mismos.

- Repositorio HTML5 y Flex para descargar (necesario mercurial):

```
1 http://code.google.com/p/babeliumproject/source/checkout
```

- Repositorio HTML5 para explorar online:

```
1 http://code.google.com/p/babeliumproject/source/browse/?name=html5
```

- Repositorio Flex para explorar online:

```
1 http://code.google.com/p/babeliumproject/source/browse/
```

A.2. Similitudes entre MXML y CSS3

A.2.1. Popiedades de estilos MXML en CSS3

La siguiente tabla refleja la comparativa o lista de similitudes entre la forma en la que se estiliza y estructuran los contenedores y/o componentes en ambos lenguajes:

MXML Style	CSS3	Comentarios
backgroundAlpha	background-color: rgba(x,y,z, a)	Aún no soportado en algunos navegadores
backgroundColor	background-color	
backgroundImage	background-image	Sustituir Embed(..) por url(..)
backgroundImageFillMode	background-repeat	
borderColor	border-color	
cornerRadius	border-radius	
borderSides	-	No soportado
borderSize	border-width	
borderStyle	border-style	
borderVisible	-	Si es false, establecer border: none;
borderWeight	border-width	
color	color	
fontFamily	font-family	
fontSize	font-size	
fontWeight	font-weight	
height	height	
minHeight	min-height	
minWidth	min-width	
paddingBottom	padding-bottom	
paddingLeft	padding-left	
paddingRight	padding-right	
paddingTop	padding-top	
roundedBottomCorners	border-bottom-radius: x;	x = 0 si es falso.
roundedTopCorners	border-top-radius: x;	x = 0 si es falso
verticalAlign	vertical-align	
width	width	

Cuadro A.1: Propiedades de estilos MXML vs CSS3

A.2.2. Selectores entre Flex y CSS3

Los *selectors* (selectores en castellano) son propiedades de CSS3/JavaScript que sirven para seleccionar objetos concretos de la jerarquía HTML en estados concretos. Usualmente suelen utilizarse para cambiar el estilo/diseño/estructura de la web. Por ejemplo: el cambio de color y subrayado de un enlace cuando un usuario pasa el ratón por encima. Aunque el ejemplo mencionado, ya está implementado por defecto en CSS3, el mismo ejemplo sirve para el cambio de color de un botón al pasar el ratón por encima.

Sin embargo, en Flex esto no se realiza mediante selectores, sino mediante propiedades especiales de los estilos MXML, con los sufijos *-Over* y *-Down* para el ejemplo:

```
1   colorOver:#2d89a4;
2   colorDown:#666666;
3   underlineOver:true;
4   underlineDown:false;
```

Que, traducido a CSS3+Selectors, quedaría algo así:

```
1 a {
2   color: #666666;
3   text-decoration: none;
4 }
5
6 a:hover {
7   color: #2d89a4;
8   text-decoration: underline;
9 }
```

El selector `:hover` hace referencia a cuando el usuario posiciona el ratón encima del objeto (en este caso enlace), y cambia el estilo del enlace. Cuando abandona el foco del objeto, vuelve a su estado normal.

En HTML5 existen, entre otros, los siguientes selectors:

#id Hace referencia a un componente en HTML5 con ID `id`, ej:

```
1 #mainContainer
```

***** Selecciona todos los elementos. Pueden seleccionarse de forma jerárquica, ej:

```
1 #mainContainer > *
```

Seleccionaría todos los componentes que hay dentro del contenedor `mainContainer`

:visited . Selecciona el link cuando ya ha sido visitado:

```
1 a:visited
```

:hover . Cuando el ratón se posiciona encima del objeto

:focus . Cuando el objeto coge el foco, por ejemplo: cuando se activa un campo de un formulario.

:first-child . Primer componente hijo de un elemento, ej:

```
1 #mainContainer:first-child
```

:last-child . Similar al anterior, pero con el último elemento

:nth-child(n) . Similar a los anteriores, selecciona el hijo número N

:enabled . Cuando está habilitado el componente (:disabled para el caso contrario).

:checked . Hace referencia a cuando un elemento de un formulario (por ejemplo, un checkbox) está seleccionado.

A.3. Estructura de carpetas y archivos del proyecto

La finalidad de esta sección reside en mostrar la estructura interna del proyecto, por directorios y carpetas, para facilitar la adaptación al proyecto a un desarrollador nuevo; es decir, suponiendo que se ha leído esta memoria y comprendido el funcionamiento y estructura de la aplicación, tanto en el cliente como en el servidor, esta sección desglosa el contenido del repositorio en jerarquías de ficheros y carpetas:

Figura A.1: Estructura de carpetas del proyecto

api: El directorio */api* es donde se aloja la API de Babelium Project. En su directorio raiz se encuentra el gateway **rest.php** y en el subdirectorio *services* todas las clases PHP que manejan la información de la base de datos.

config: Guarda una plantilla (Config.php.template) de la configuración de la aplicación. Hay que renombrarla a **Config.php** y editar la información en su interior (por

ejemplo: usuario, host y password de la base de datos) para poner la aplicación en funcionamiento.

core: Contiene los *managers* o *helpers* que controlan toda la gestión de datos y contenidos en el servidor: ModuleLoader, WidgetLoader y Session Manager.

demo: Es un directorio ajeno a la página web, pero contiene *demos* de prueba realizados durante la primera fase del proyecto, la investigación de Flex y las posibilidades de migración a HTML5.

locale: Contiene los ficheros de traducciones para los idiomas soportados por Babelium Project (en estos momentos, en_US, es_ES y eu_ES).

modules: En su carpeta raiz residen todos los módulos de la aplicación (Home, Practice, Evaluation, Configuration, Register, Auth...).

themes: El servidor tiene un gestor de *themes* (o temas) configurado en Config.php. Sólamente hay uno creado, el que utiliza Babelium por defecto, y en su interior se aloja **todo** lo relativo al cliente de la aplicación: estilos CSS, script JavaScript, Cairngorm, imágenes, e incluso las plantillas `.tpl` que después parseará Smarty.

util: Directorio genérico para utilidades del proyecto. Dentro se encuentra el plugin Smarty `i18n`, el reproductor de vídeo, interfaces que implementan algunas clases (iSingleton, iModule e iWidget), un script SQL con la última versión de la base de datos...

widgets: Ordenados por subcarpetas, se encuentran los widgets de cada módulo.

A.4. Implementación del framework Cairngorm

En la sección 5.1.1 se explicó el funcionamiento básico de Cairngorm, que queda reflejado en la siguiente figura, mostrada también en esa misma sección.

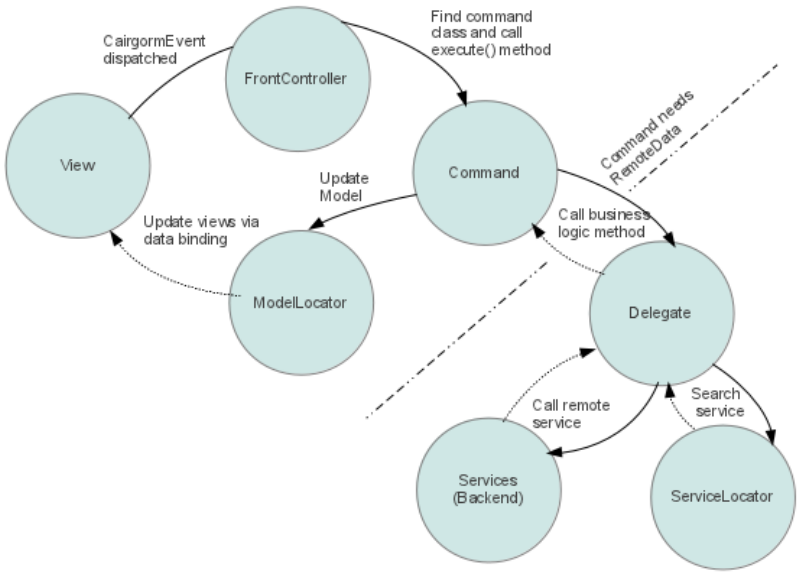


Figura A.2: Arquitectura de Cairngorm MVC

Como se puede observar en la figura, en la acción desde que el usuario pulsa en un botón hasta que se actualiza la información de la página, intervienen: `eventos`, `Event Dispatcher`, `Front Controller`, `comandos`, delegados, `Service Locator` y `HTTP Services`.

La estructura interna de Cairngorm sólo implementa el patrón MVC, el patrón delegate que implementan las clases "delegate" es una adición de Babelium Project para una mejor delegación de tareas. Todo lo demás, sin embargo, forma parte de la implementación Cairngorm, tanto en Flex como en JavaScript. En la figura A.3 se puede observar un diagrama de clases de Cairngorm en JavaScript:

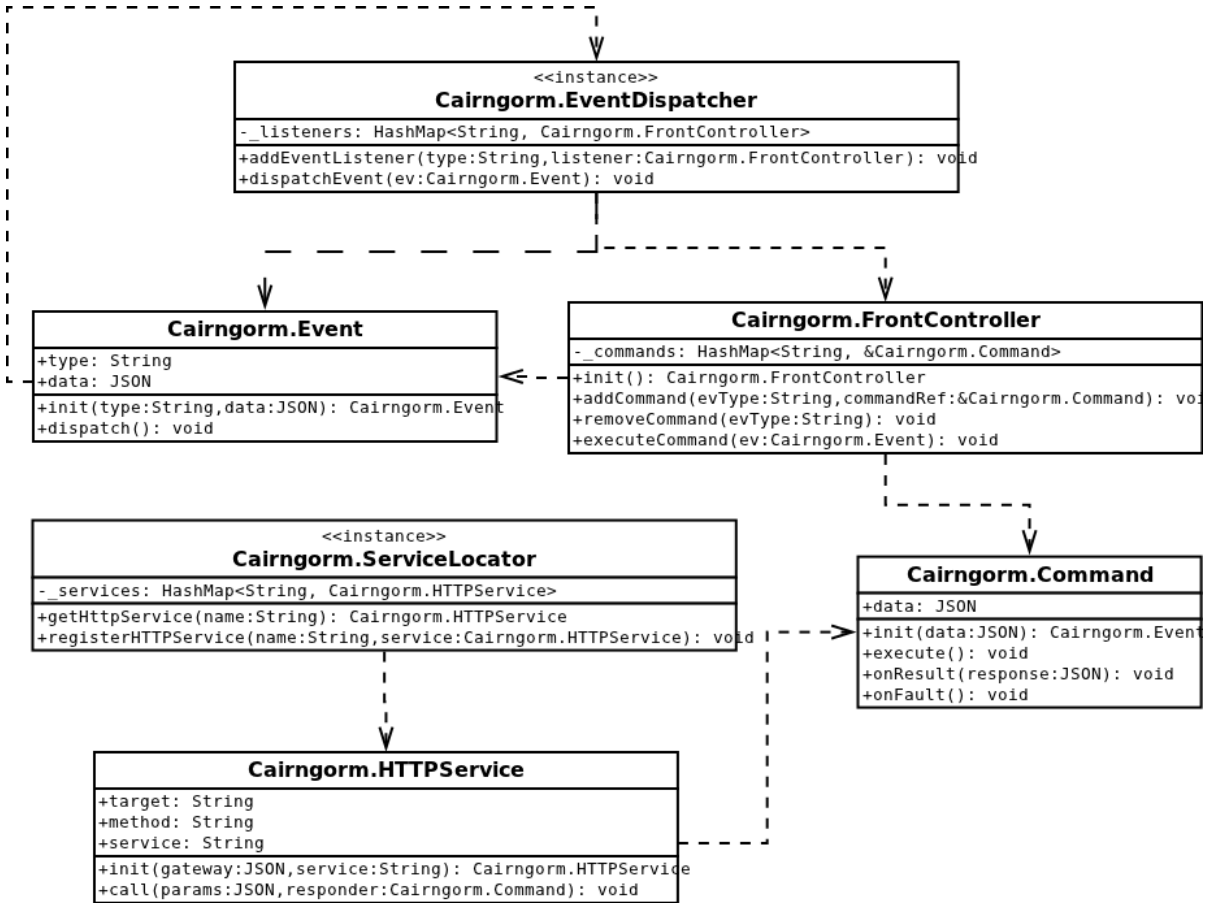


Figura A.3: Diagrama de clases de Cairngorm JS

1. Cuando se llama al método `dispatch` de un evento, éste invoca la instancia única del *EventDispatcher* y llama a su método `dispatchEvent`, enviándose al propio evento como parámetro.

2. *EventDispatcher* busca en su tabla hash (`_listeners`) cual de los controladores está capacitado para atender a dicho evento (mirando su tipo). Cuando lo encuentra, llama al método `executeCommand` del controlador, enviando el mismo evento como parámetro.

3. El *FrontController* encargado de gestionar el evento, busca en su tabla hash (`_commands`) la clase de comando asociado a ese tipo de evento, crea una instancia nueva de comando pasándole el campo `data` del evento como parámetro, y llama al método `execute` del comando.

4. Supongamos, para el ejemplo, que el comando requiere de información externa. Para ello llamaría a un delegado (que no forma parte de la arquitectura de Cairngorm pero estarían creados para Babelium Project) pasándose el mismo comando como *responder*.

5. El delegado, mediante el *ServiceLocator* buscaría el *HTTPService* necesario para llevar a cavo la acción, y mediante el método `call` del *HTTPService* obtendría el resultado del servicio externo, asignando el comando como *callback*.

A.5. Implementación de la infraestructura en el cliente

Como hemos mencionado a lo largo de la memoria, la parte del cliente se traduce en una serie de clases que implementan el patrón MVC que Cairngorm (traducción a JavaScript en nuestro caso) ofrece. Entre ellas están los comandos, los eventos, los delegados y el controlador (aunque podría tener varios, Babelium dispone de un único controlador):

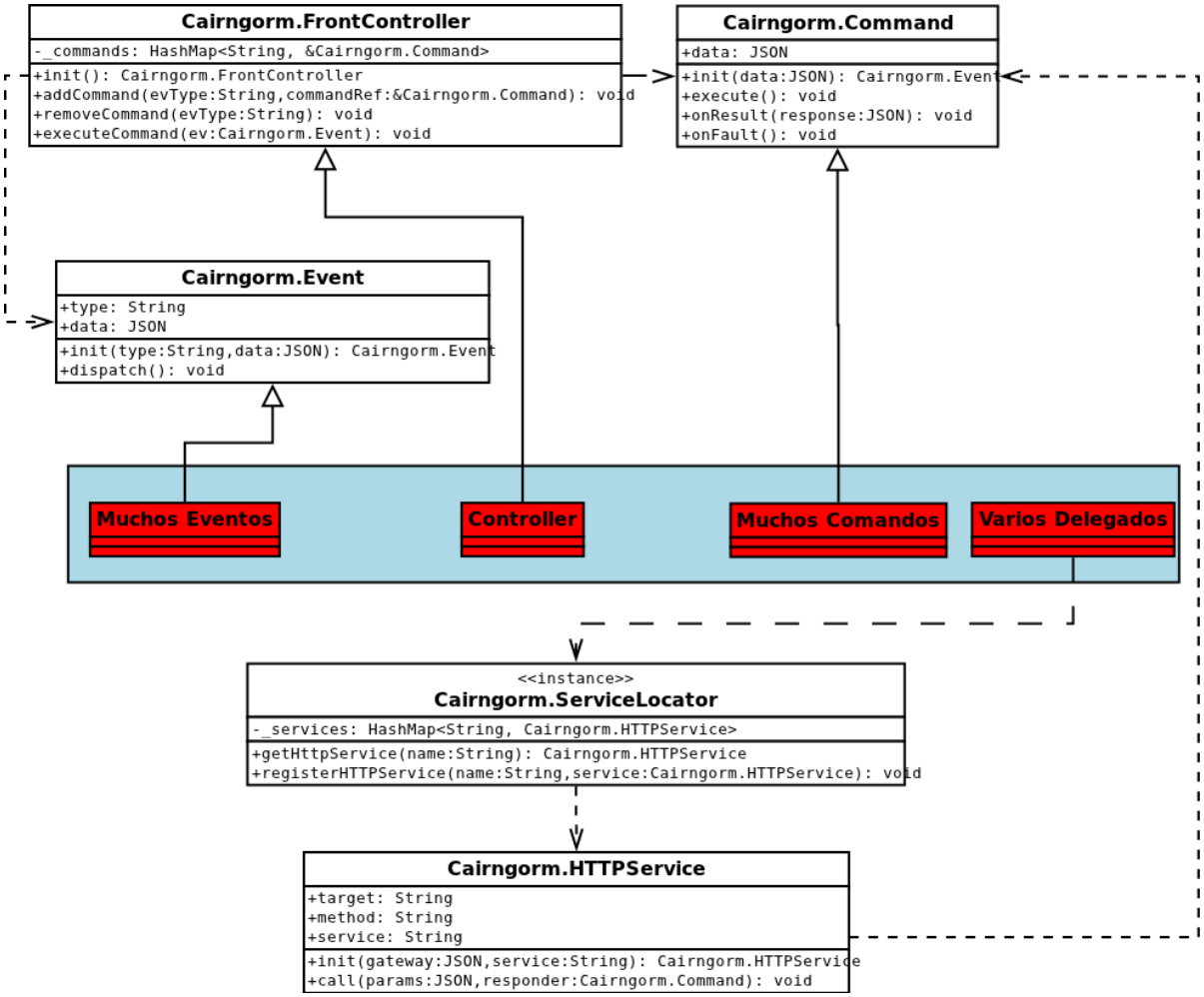


Figura A.4: Diagrama de clases de Babelium JS

Además de los elementos que implementan el patrón MVC+Delegate, la librería *babelium.js* también dispone de algunas clases *helpers* para ofrecer utilizades a los comandos:

BP.CMS: *Content Management System* (sección 8.2.2), encargado de ofrecer distintas funciones útiles para la inserción y eliminación de contenido en la web.

BP.SM: *State Manager* (sección 8.3.3), encargado de guardar y gestionar los cambios de estado en la web, para ofrecer una navegación estática y dinámica al mismo tiempo, sin recargos de página pero con actualización de URL y situación.

BP.EM: *Exercise Manager* (sección 8.6.4), encargado de gestionar las acciones relativas al reproductor de vídeo. Carga y descarga de ejercicios, búsqueda de subtítulos, e implementación del algoritmo basado en cuepoints (**??**) para la gestión de acciones en el reproductor.

BP.Services: Clase que implementa los métodos y protocolos de conexión directa con la API (mencionada en la sección 8.3.1). Se utiliza para llamadas directas a la API que no requieren de respuesta alguna, de carga de contenidos adicionales o de alteración de la UI de la web; por ejemplo, llamadas a la API para: obtener los subtítulos de un vídeo, para guardar una evaluación, para publicar un ejercicio grabado..

Ejemplos de implementación de comandos, eventos y delegados están disponibles en la sección 7.5 y anexo B.3. A continuación se muestra explicación un poco detallada de cómo funcionan los helpers y tu real utilidad.

A.5.1. BP.CMS

Es una instancia única invocable desde cualquier comando con el propósito de conseguir que la web cargue contenido dinámicamente de una forma elegante, mediante efectos (*fadeIn*, *fadeOut*).. La figura A.5 refleja un diagrama de clase con los métodos principales.

Los métodos `prepare-` y `inner-` son los principales y más usados de esta clase (que internamente utilizan los otros mencionados). A modo de ejemplo se muestra el código de un comando que muestra el módulo *home*:

```
1 var ViewHomeModuleCommand = Cairngorm.Command.extend(
2 {
3   execute : function ( )
4   {
5     var _this = this;
6
7     BP.CMS.prepareMainContent("home", function ()
8     {
```

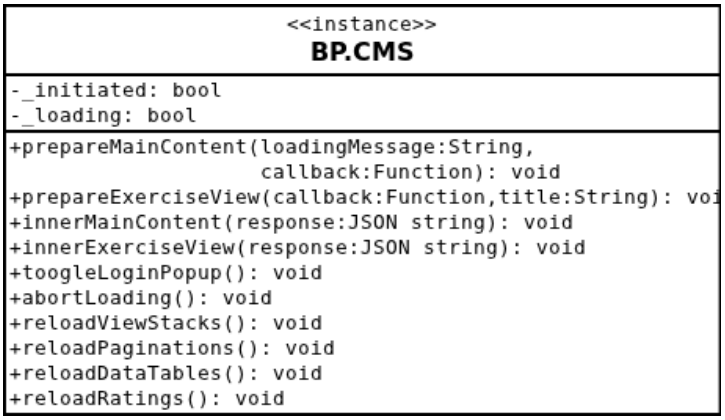


Figura A.5: Babelium JS: CMS

```
9  P.HomeDelegate.viewHomeModule(_this);
10     });
11   },
12
13   onResult : function ( response )
14   {
15     BP.CMS.innerMainContent(response);
16   },
17
18   onFault : function ()
19   {
20 / Error
21   }
22 });
```

1. Primero se ejecuta el método `execute` del comando, y se llama al `prepare main content` del CMS. Este método elimina todo el contenido de la sección principal, y cuando termina, llama a la función *callback* pasada por parámetro, que a su vez llama a un delegado. Recordamos que un delegado lanzará una petición al gateway de contenidos y devolverá el contenido a `onResult`, en caso de una respuesta correcto, u `onFault` en caso de error.

2. Cuando el comando recibe la respuesta en `onResult` (suponiendo que ha ido bien), se invoca de nuevo al CMS para que inserte el nuevo contenido que ha recibido.

A.5.2. BP.SM

Instancia única, invocable desde cualquier comando, que contiene 2 métodos principales (pushState y onPopState), 3 métodos *helpers* (at, action, params) y un método

para devolver el estado actual de la aplicación:

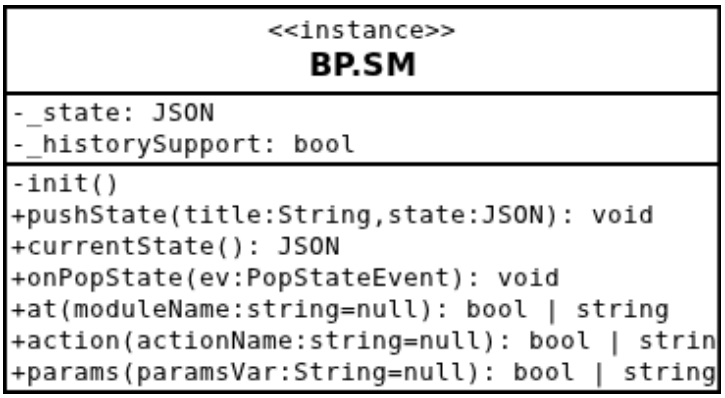


Figura A.6: Babelium JS: SM

pushState(title, data) : guarda el estado como último estado que la página ha visitado, siendo `title` el título que aparecerá en el historial y `data` el objeto JSON que se guardará a modo de estado y se representará en el cambio de URL, ejemplo:

```
1 BP.SM.pushState('home', {module: 'home', action: 'activity'});
```

Se traducirá en la URL a *?module=home&action=activity* y se guardará una entrada en el historial de navegación del navegador, con dicha URL y el título "home".

onPopState(ev) : callback del evento OnPopStateEvent que será llamado cuando el usuario pulse atrás o adelante en el navegador (o alguna función haga retroceder o avanzar la página). Internamente hará que la página recargue el último estado que ha visitado y su contenido asociado.

at, action, params : son métodos *helpers* que sirven para comprobar en que estado se encuentra la aplicación y cargar distintos estados en consecuencia. Si estos métodos no reciben ningún parámetro, devuelven el parámetro (module, action o params respectivamente) del último estado de la aplicación. En caso contrario, por ejemplo que al método `at` se le pase un parámetro, éste devolverá `true` o `false` en función de si el módulo pasado por parámetro y el del último estado coinciden. Ejemplo:

```
1 BP.SM.pushState('home', {module: 'home', action: 'activity'});
2
3 BP.SM.at(); // devuelve 'home'
```

```
BP.SM.action(); // devuelve 'activity'
BP.SM.params(); // devuelve null
BP.SM.at('home'); // devuelve true
BP.SM.at('practice'); // devuelve false
BP.SM.action('view'); // devuelve false
```

currentState , simplemente devuelve el estado actual de la aplicación.

A.5.3. BP.Services

Clase que hace de gateway con la API. Nada más inicializarse la aplicación automáticamente inicializa la autenticación mediante tokens con la API (siguiendo el protocolo de la sección 8.3.1). Una vez cargada por primera vez la aplicación, esta instancia única sólo tiene un método útil:

```
send = function(secured, method, parameters, callback) {..}
```

Siendo:

secured un valor booleano indicando si usa http o https

method el método de la API que se desea invocar

parameters los parámetros del método de la API

callback la función que queremos que reciba el resultado

A modo de ejemplo. se muestra el método `saveResponse` de *BP.EM*, que registra una grabación en la base de datos:

```
// Save Response
this.saveResponse = function ()
{
  var responseThumbnail = "nothumb.png";
  var subtitleId = this.cueManager.currentSubtitle();

  var duration = this.bpPlayer.duration();

  // Prepare an AJAX call to the appointed service
  var parameters = {
    'userId' : null,
```

```
12      'exerciseId' : instance.exerciseId,
13      'fileIdentifier' : instance.recordedFilename,
14      'isPrivate' : true,
15      'thumbnailUri' : responseThumbnail,
16      'source' : 'Red5',
17      'duration' : duration,
18      'addingDate' : null,
19      'ratingAmount' : 0,
20      'characterName' : instance.selectedRole,
21      'transcriptionId' : 0,
22      'subtitleId' : subtitleId
23    };
24
25    BP.Services.send(false, "saveResponse", parameters, saveResponseCallback);
26  };
```

A.6. Implementación de la infraestructura en el servidor

Como se mencionó en el apartado 8.2.1, el servidor está formado por una jerarquía de módulos y widgets, y dos *loaders* que los cargan. En la siguiente figura puede observarse su diagrama de clases:

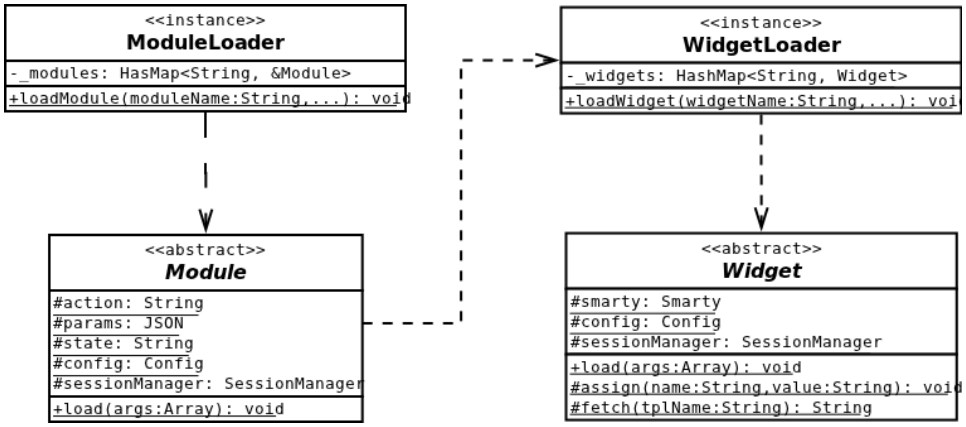


Figura A.7: Diagramas de la carga de Módulos y widgets

1. ModuleLoader tiene una tabla hash en su interior, con claves nombre de tipo String y como valores tipos de clases que heredan de `Module`.

2. De la misma manera, WidgetLoader tiene una tabla hash de Strings y clases que heredan de `Widget`

3. Uno de los gateways de la aplicación recibe por URL el módulo + acción + parámetros + estado de la aplicación, y se los envía a ModuleLoader (mediante el método `loadModule`):

```
1 ?module=home&action=activity&params=&state=
```

```
1 ModuleLoader::loadModule("home", "activity");
```

4. ModuleLoader busca el módulo con nombre **home** en su tabla Hash, y, suponiendo que existe, obtiene su clase `Module` (MHome) asociada y llama a su método `load`.

```
MHome::load(array("home", "activity"));
```

5. La clase MHome llama a el método `load` de su clase padre (Module), lo que hace que las variables protegidas obtengan los valores apropiados.

6. MHome analiza la acción y parámetros que ModuleLoader le ha enviado, y en función de ellos, realiza la carga de un widget u otro.

7. En este caso, carga el widget Activity, y para ello antes busca la información necesaria en los servicios y se la envía a widget loader:

```
$home = new Home();
$received = $home->usersLatestReceivedAssessments();
$given = $home->usersLatestGivenAssessments();
WidgetLoader::loadWidget("activity", $received, $given);
```

8. WidgetLoader busca y carga el widget "activity" y llama a su método `load` pasándole los parámetros:

```
WActivity::load(array("activity", $received, $given));
```

9. El widget de activity llamará al método load de su clase padre (Widget) y eso dará valor adecuado a sus variables protegidas.

10. El widget activity hará las asignaciones pertinentes a la clase Smarty (utilizando los métodos del padre) para luego procesar las plantillas:

```
// Prepare template
self::assign("received", $received);
self::assign("given", $given);
```

11. Por último, el widget devolverá el procesado de la plantilla Smarty:

```
return self::fetch("home/LatestUserActivity.tpl");
```

Apéndice B

Guía para desarrolladores: Cómo migrar un módulo

Contents

A modo de ejemplo, explicaré la migración del módulo Evaluation que, al igual que todos los módulos, contendrá lo siguiente:

- Lógica de generación de la representación (PHP, Smarty, TPL files, Services)
- Representación gráfica (HTML5+CSS3)
- Carga dinámica de widgets del módulo (JS+Cairngorm+JQuery)
- Navegación dinámica por las secciones (acciones) del módulo (HTML5, JS, JQuery)

Para ello, de forma genérica para cualquier otro módulo, se seguirán los siguientes pasos:

1. Creación de la lógica de representación de la sección principal del módulo
2. Creación de la representación principal del módulo
3. Creación de los eventos/comandos que realizarán acciones sobre el módulo principal

Por cada acción disponible en el módulo:

5. Adición de la lógica de representación de la acción requerida
6. Creación de la representación (si la tuviera) de la acción
7. Creación de los eventos/comandos que permitan cambiar a la acción en cuestión
8. Creación de los eventos/comandos que realizarán otras acciones sobre el estado actual del módulo

Y para migrar el módulo correctamente, se hará siguiendo la lógica utilizada en la versión Flex, por lo que a lo largo del documento surgirán numerosas citas a ficheros y líneas de código de dicha versión. Se puede observar el código online o descargarse el respositorio:

```
1 http://code.google.com/p/babeliumproject/source/browse/
```

Se presupone que se ha leído y entendido la estructura de la aplicación, explicada en la sección 8.2, antes de continuar. Así mismo, se presupone que se está trabajando sobre el código del repositorio, por lo que los paths de carpetas serán relativos al root / del proyecto.

```
1 http://code.google.com/p/babeliumproject/source/browse/?name=html5
```

B.1. Creación de la lógica de representación de la sección principal del módulo

Esta sección no tiene, en principio, similitud con la lógica utilizada en Flex, ya que en Flex toda la aplicación está encapsulada en el mismo programa en el cliente, mientras que en HTML5 existe parte cliente para la navegación y parte servidor para la generación de contenidos. Es por ello que sigue un patrón nuevo de diseño.

Primero accedemos a la carpeta */modules/* y ahí podemos encontrar todos los módulos disponibles en la aplicación. Vamos a crear uno nuevo que corresponda al módulo **Evaluation**. Creamos el archivo */modules/MEvaluation.php* extiende la clase */modules/Module.php* que implementa la interfaz `iModule.php`, lo que implica que tendrá un método `load` que cargará el módulo, recibiendo como parámetro el estado actual en el que se encuentra, para saber que acción tiene que cargar. La plantilla para cualquier módulo quedaría algo así:

```
1  <?php
2  require_once(dirname(__FILE__) . "/Module.php");
3
4  class MEvaluation extends Module
5  {
6    public static function load($args)
7    {
8      parent::load($args);
9    }
10 }
11 ?>
```

La lógica de carga de contenidos en el servidor, dispone de una clase llamada `ModuleLoader` que será la encargada de llamar a está clase cuando se pida por URL el módulo **Evaluation**. Para que dicho *loader* sepa de la existencia del módulo evaluation y cuando tiene que cargarlo, tenemos que "registrarlo":

1. Accedemos a */core/ModuleLoader.php* y buscamos el array asociativo llamado `$_module`.

2. Añadimos a dicho array una asociación con la clase de módulo que acabamos de crear. La asociación sigue el siguiente patrón:

   Nombre del módulo en la URL =>Nombre de la clase del Módulo; que en nuestro caso se traduce como:

```
1 "evaluate" => "MEvaluation"
```

3. Posteriormente añadimos una línea al comienzo del fichero importando la clase que contiene el módulo:

```
1 require_once(dirname(__FILE__) . "/../modules/MEvaluation.php");
```

Y listo. Ya tenemos una clase `MEvaluation` que se encargará de cargar el módulo evaluation en sus correspondientes estados. Siempre que se acceda por URL a cualquiera de los 2 gateways de carga de contenidos de la aplicación (*/index.php* y */modules/bridge.php*) con el nombre "evaluate" como identificador del módulo, automaticamente se llamará a la clase MEvaluation que acabamos de crear, pasándole como parámetro un array conteniendo el estado de la aplicación (o el request). Además, con el método load de la clase que hereda, automáticamente tendremos registradas en la carga del módulo las siguientes variables:

self::$action : acción requerida dentro del módulo. Si no se facilita ninguna, puede asumirse una carga standard (por defecto) del módulo.

- Ej: pueden existir acciones como "view" o "rec" en el módulo de practice

self::$params . parámetros necesarios para llevar a cavo la acción requerida por el anterior parámetro. Argumento inútil si no se ha especificado una acción válida antes.

self::$state : Un flag (por defecto a null) pensado para la carga dinámica de contenidos. Recibe el contenido del parámetro "state" enviado en la petición HTTP (AJAX), que sólo se atiende si entra por el gateway */modules/bridge.php* (*/index.php* ignora dicho parámetro). La razón de la existencia de este parámetro, es para indicar en una carga dinámica que recargue completamente el módulo (dejándolo a null), o si queremos que recargue sólo una pequeña parte de él.

- Ej: si esta variable toma el valor "min" en una llamada a /modules/bridge.php con "practice" cómo módulo a cargar, el módulo no volverá a cargar la lista de ejercicios completa, solamente retornará la información del vídeo requerido e ignorará todo lo demás.

self::$config , variable que hace referencia a la instancia de Config.php con la configuración de la aplicación

self::$sessionManager , variable que hace referencia a la instancia de SessionManager.php con la sesión y datos de la sesión del usuario

B.1.1. Carga del widget principal del módulo

Si accedemos al repositorio de Flex, podemos encontrar el código principal del módulo en la carpeta */src/modules/evaluation/** y en */src/modules/main/Body.mxml*, el contenedor principal de la aplicación, podemos encontrar la siguiente línea:

```
1 <evaluation:EvaluationContainer id="evaluationModule"/>
```

Que nos indica que *EvaluationContainer.mxml* es la clase principal del módulo evaluation, que viene a ser algo similar al *MEvaluation.php* que acabamos de crear. En el código MXML de *EvaluationContainer* se puede apreciar que lo primero que hace el módulo, es una comprobación de si el usuario está identificado o no, y en caso negativo, llevarle a una página *UnauthorizedNotice*, en caso afirmativo, a la página principal del módulo: *EvaluationMain*.

Vamos a realizar exactamente la misma operación en la versión PHP, para ello añadimos las siguientes líneas al método `load` que acabamos de crear en la clase *MEvaluation*:

```
1 $loggedIn = self::$sessionManager->isLoggedIn();
2
3 if ( !$loggedIn )
4   return WidgetLoader::loadWidget("Unauthorized");
```

La lógica de negocio contiene *managers* o *helpers* para la carga de contenidos y realización de acciones habituales, como son en este caso `SessionManager` y `WidgetLoader`.

1. La primera se encarga de gestionar la sesión del usuario, se inicializa automáticamente en cada llamada a la parte del servidor e identifica al usuario. Es por ello que podemos usarla desde nuestro módulo sin preocupaciones para saber si el usuario este identificado como un usuario registrado, o no.

2. La segunda clase, WidgetLoader, se encarga de cargar los contenidos de la página web, en forma de plantillas TPL (que contienen código PHPSmarty + HTML5). En el ejemplo anterior, le estamos diciendo que cargue la plantilla `Unauthorized` si el usuario no está identificado.

Ahora accedemos al archivo principal del módulo, *EvaluationMain.mxml* para ver como está estructurado. Primero, vamos a ignorar por completo el contenido ActionScript, y centrarnos sólo en la parte visual (MXML) que carga. El módulo evaluation requiere un poco de ActionScript para cargar dinámicamente las secciones, pero entraremos en ello un poco más tarde.

El primer contenido que aparece es la cabecera de la sección:

```
1 <s:BorderContainer styleName="sectionInfoBox" [\global] [\let] [\OT1\textellipsis] [.\k
```

En la que al final, podemos encontrar un *TabBar* que nos permitirá cambiar entre las secciones (acciones, estados..) del módulo:

```
1 <s:TabBar dataProvider="{evaluationOptionsViewStack}"
2     skinClass="skins.AccountTabBarSkin"
3     cornerRadius="8"/>
```

Sin embargo, cuando buscamos el dataProvider `evaluationOptionsViewStack`, vemos que se encuentra vacío:

```
1 <mx:ViewStack id="evaluationOptionsViewStack"
2     width="100%"
3     height="100%"
4     creationPolicy="all"
5     resizeToContent="true"
6     change="onEvaluationTabChange(event)" />
7 </mx:ViewStack>
```

Esto es así, porque dependiendo de niveles de privilegios del usuario (común, administrador..) se cargarán unas pestañas u otras, y si nos fijamos ligeramente en el código ActionScript, en el método `preinitializeHandler` se cargan las pestañas correspondientes, en función de los datos del usuario:

```
1 if (CONFIG::restrictedEvaluation)
2   if (dataModel.loggedUser.isAdmin == true)
3     // Cargar todas las pesta\penalty \@M \hskip \z@skip \unhbox \voidb@x \bgroup \le
4   else
5     // Cargar pesta\penalty \@M \hskip \z@skip \unhbox \voidb@x \bgroup \let \unhbox
6 else
7     // Cargar s\penalty \@M \hskip \z@skip \unhbox \voidb@x \bgroup \let \unhbox \voi
```

Con lo cual, obviando un poco el contenido dentro de los `ifs`, vamos a seguir su misma lógica, y pedirle al WidgetLoader que nos cargue un TabBar, y para que lo haga correctamente vamos a enviarle los datos que necesita. Añadimos la siguiente línea al MEvaluation.php:

```
1 $content = WidgetLoader::loadWidget("EvaluationTabBar",
2 elf::$config->restrictedEvaluation,
3 elf::$sessionManager->getUserData());
```

`$content` es una variable creada para guardar todo el contenido que el módulo va a cargar. Al final de la carga añadiremos un `return $content;` para que imprima correctamente todo lo necesario:

self::$config-¿restrictedEvaluation hace referencia a la variable restrictedEvaluation a través del helper Config.php, que guarda toda la config de la aplicación.

self::$sessionManager-¿getUserData() es una función que devuelve un objeto con los datos del usuario.

Y por ahora, ya tenemos la creación de la página principal del módulo Evaluation, vacía en un principio, solamente con las pestañas, que cargarán las subsecciones dependiendo cuál elijan, pero más adelante se entrará en detalle de esta parte.

El siguiente paso, será la creación de los widgets que el widget loader debe cargar: **Unauthorized** y **EvaluationTabBar**.

B.1.2. Creación de los widgets contenedores del módulo

En el código del módulo le hemos pedido que nos cargue 2 widgets: **Unauthorized** y **EvaluationTabBar**. Vamos a proceder a crearlos.

Para ello, accederemos primero a la carpeta donde se guardan los Widgets, que es: */widgets/*, y crearemos dentro los 2 ficheros correspondientes, que extienden de `Widget` que a su vez implementa la interfaz `IWidget`, cuya plantilla general sería algo así:

```
1 <?php
2 require_once(dirname(__FILE__) . "/Widget.php");
3
4 class WExample extends Widget
```

```
5 {
6   public static function load($args)
7   {
8     parent::load($args);
9   }
10 }
11 ?>
```

Al llamar al método `load` del padre, automáticamente se generan las siguientes variables útiles:

self::$config , variable que hace referencia a la instancia de Config.php

self::$sessionManager , variable que hace referencia a la instancia de SessionManager.php

self::$smarty , variable que hace referencia a la instancia de Smarty (el gestor de plantillas)

Además, la clase Widget provee de los siguientes métodos estáticos:

self::assign(name, value) : permite asignar un valor a una variable (name) de Smarty

self::fetch(tplName) : devuelve en un string el contenido de parsear la plantilla pasada por parámetro

Explicado lo anterior, se procederá a la creación de widgets en los siguientes subapartados.

Creación del widget Unauthorized

El widget unauthorized es bastante simple, no tiene que mostrar más que el mensaje "No autorizado", por lo que no necesita de ningún parámetro extra. Dicho esto, el widget quedaría de esta forma, siguiendo la plantilla explicada anteriormente:

```
1 class WUnauthorized extends Widget
2 {
3   public static function load($args)
4   {
5     parent::load($args);
6     return self::fetch("Unauthorized.tpl");
7   }
8 }
```

Dado que el este widget será usado por más módulos, se situará directamente de la carpeta */widgets/* (no en ninguna subcarpeta, destinadas a los widgets propios de cada módulo).

Creación del widget EvaluationTabBar

EvaluationTabBar tampoco tiene gran complicación, simplemente hay que tener en cuenta dos variables para saber qué mostrar. En lugar de hacer las comprobaciones en PHP y cargar plantillas distintas, vamos a hacer que Smarty haga las comprobaciones dentro de una misma plantilla, por ello, asignaremos a variables smarty los datos necesarios, quedando de la siguiente forma:

```
1 class WEvaluationTabBar extends Widget
2 {
3   public static function load($args)
4   {
5     parent::load($args);
6
7     self::assign("restrictedEvaluation", $args[1]);
8     self::assign("isAdmin", $args[2]);
9
10    return self::fetch("evaluation/EvaluationTabBar.tpl");
11  }
12 }
```

Nota: las plantillas Smarty tienen la extensión `.tpl` y están situadas en */themes/babelium/templates/*.

Los widgets reciben todos los parámetros encapsulados en el array $args, siendo el primer elemento del array (posición 0) el propio nombre del widget, es por ello que los parámetros enviados desde el módulo se encuentran en la posición 1 y 2 respectivamente.

Añadir los widgets creados al WidgetLoader

Los widgets hay que registrarlos, al igual que hicimos con los módulos, al */core/WidgetLoader.php*. Hay que añadir los imports primero:

```
1 require_once(dirname(__FILE__) . "/../widgets/evaluation/WEvaluationTabBar.php");
2 require_once(dirname(__FILE__) . "/../widgets/WUnauthorized.php");
```

Acto seguido, añadirlos a la tabla hash de widgets disponibles, `$_widget`:

```
"Unauthorized" => "WUnauthorized",
"EvaluationTabBar" => "WEvaluationTabBar"
```

Y listo, el siguiente y último paso de la generación de contenido principal del módulo, será crear las plantillas `.tpl` que le hemos pedido a Smarty que cargue.

B.2. Creación de los contenidos principales del módulo

Primero, vamos a crear ambos widgets en la carpeta correspondiente a las plantillas, que es: */themes/babelium/templates/*, y crearemos dentro los 2 ficheros:

- ./Unauthorized.tpl
- ./evaluation/EvaluationTabBar.tpl

Toda sección/acción que cargue un módulo, debe ir dentro de su correspondiente `<section>`, incluyendo el título de la sección dentro de la subetiqueta `<header>`, esto es así debido al funcionamiento del *Content Management System*, explicado en el anexo 8.2.2. Cómo ejemplo simple, el widget de **Unauthorized** quedaría algo así:

```
<section>
  <header>
    <h1>Unauthorized</h1>
  </header>
</section>
```

Ahora mismo, si accedemos al módulo evaluation sin identificarnos, deberíamos ver el aviso que puede leerse claramente en el código de arriba.

Procederemos a la creación de la siguiente plantilla, un poco más compleja.

B.2.1. Creación del la plantilla de EvaluationTabBar

Vamos a entrar ahora con la plantilla de EvaluationTabBar. Tenemos que tener en cuenta varias cosas:

1. Cargar unas pestañas u otras en función de la configuración y del nivel de usuario
2. En html5 no existe como tal el componente TabBar, así que hay que crear uno. Para ello disponemos de algunas funciones JavaScript y la plataforma Cairngorm para implementarlo.
3. En Flex los TabBar tienen el contenido de cada pestaña ya integrado y compilado, para esta versión, cargaremos el contenido de cada pestaña dinámicamente mediante secuencias de Cairngorm.

Definición de los eventos Cairngorm

Las pestañas, fijándonos en el código MXML de *EvaluationMain.mxml161*:

```
1 evaluationOptionsViewStack.addChild(waitingAssessmentBoxNavContent);                    eval
```

Con lo cual, los eventos podrían ser los siguientes, uno por cada pestaña:

```
1 EvaluationEvent.VIEW_WAITING_ASSESMENTS
2 EvaluationEvent.VIEW_CURRENTLY_ASSESSED_TO_USER
3 EvaluationEvent.VIEW_CURRENTLY_ASSESSED_BY_USER
```

Creando la plantilla HTML+CSS

Teniendo en cuenta las condiciones vistas anteriormente, se mostrarán todas las pestañas sí el usuario es un administrador o si no está en modo restrictivo; y, en caso contrario, sólo la segunda.

A continuación podremos ver una plantilla que facilitará las pestañas en función de las variables, y que permitirá cambiar de contenido de una a otra mediante eventos. Entre llaves ({ y }) se puede observar código Smarty que carga una o varias pestañas, en función de las variables `isAdmin` y `restrictedEvaluation` que le hemos asignado en el

punto B.1.2; dentro de los `a href` se puede observar código JavaScript para disparar los eventos que cambiarán de una pestaña a otra, y el resto lo compone el código HTML que forma la lista de pestañas. Por último, la etiqueta `class` del contenedor `aside` hace referencia a una clase CSS3 especial que permite crear un bloque que se alineará a la derecha, mediante las Flexible Boxes de HTML5, haciendo que las pestañas queden alineadas a la derecha:

NOTA: más adelante en este mismo documento se detalla cómo crear y programar la secuencia Cairngorm para que al pulsar en los botones se cambie de una pestaña a otra.

```
<aside id="TabBarContainer" class="HBox hend">
  <ul id="evaluationTabBar" class="HBox">
{if $isAdmin || !$restrictedEvaluation}
    <li>
     <a href="javascript:new EvaluationEvent(
          EvaluationEvent.VIEW_WAITING_ASSESMENTS).dispatch();">
      Waiting for evaluation
     </a>
    </li>
    <li>
     <a href="javascript:new EvaluationEvent(
          EvaluationEvent.VIEW_CURRENTLY_ASSESSED_TO_USER).dispatch();">
      Currently assessed to user
     </a>
    </li>
    <li>
     <a href="javascript:new EvaluationEvent(
          EvaluationEvent.VIEW_CURRENTLY_ASSESSED_BY_USER).dispatch();">
      Currently assessed by user
     </a>
    </li>
{else}
    <li>
     <a href="javascript:new EvaluationEvent(
          EvaluationEvent.VIEW_CURRENTLY_ASSESSED_TO_USER).dispatch();">
      Currently assessed to user
     </a>
    </li>
{/if}
 </ul>
</aside>
```

Dándole un poco de estilo

Por último, con un poquito de magia CSS3, vamos a conseguir que las pestañas queden elegantes. Para ello está preparado en */themes/babelium/css/main.css* las clases:

aside#tabBarContainer para el contenedor de las pestañas.

aside#tabBarContainer > ul > li para el estilo de cada pestaña.

aside#tabBarContainer > ul > li:hover para resaltar la pestaña al pasar el ratón por encima.

```
aside#tabBarContainer
{
  width: 100%;
}

  aside#tabBarContainer > ul > li
  {
    padding: 7px 15px 5px 15px;
    margin-right: 3px;

    font-weight: bold;

    border-bottom: 1px solid #38babc;
    border-left: 1px solid #38babc;
    border-right: 1px solid #38babc;

    /** background gradient **/
    background: linear-gradient(top, white, #E2EFF3);

    /** rounded borders **/
    border-bottom-left-radius:6px;
    border-bottom-right-radius:6px;
  }

  aside#tabBarContainer > ul > li:hover
  {
    cursor: pointer;

    /** another background gradient **/
    background: linear-gradient(top, white, #86BCCE);
    text-decoration: none;
    text-shadow: 1px 1px 1px rgba(0,0,0,.2);
  }
```

Y ya tenemos la generación de pestañas terminada. En los próximos pasos se procederá a explicar como cargar las acciones y el contenido de cada una de ellas.

B.3. Eventos y comandos para la navegación dinámica

Este paso se realiza en la parte "cliente" de la aplicación. Toda la parte de implementación de Cairngorm, eventos y comandos se realiza en la librería `babelium.js`,

cuyos ficheros están situados en */themes/babelium/js/babelium/**. Dichos ficheros se compilarán y generarán el fichero `babelium.js` (usando la utilidad ANT).

NOTA: Todos los paths de aquí en adelante harán referencia a dicha carpeta.

B.3.1. Creación de los eventos

Los eventos se sitúan en la carpeta */events/*, y son clases JavaScript que extienden la clase `Cairngorm.Event`. Vamos a acceder a dicha carpeta, y vamos a crear el evento *EvaluationEvent.js* que hemos definido en la sección B.2.1, que además tiene la estructura idéntica que el evento de Flex (/src/events/EvaluationEvent.as):

```
// Event
var EvaluationEvent = Cairngorm.Event.extend(
{
  // Constructor del evento
  init : function ( type, exercise, report, score )
  {
    this._super(type, {
      "evaluation" : evaluation,
      "responseId" : responseId,
      "sortField" : sortField,
      "pageNumber" : pageNumber
    });
  }
});

// Constants
EvaluationEvent.VIEW_WAITING_ASSESMENTS = "viewWaitingAssesments";
EvaluationEvent.VIEW_CURRENTLY_ASSESSED_TO_USER = "viewAssessedToUser";
EvaluationEvent.VIEW_CURRENTLY_ASSESSED_BY_USER = "viewAssessedByUser";
```

El evento EvaluationEvent de Flex aún tiene más cosas, pero vamos a ir migrando sólo las que vamos necesitando. El segundo parámetro del método `_super` (padre), se guardará como objeto (JSON) `.data` del evento, que a su vez será enviado al comando que se ejecute a continuación.

Ahora, vamos a registrar dichos tipos de eventos en el controlador, que al igual que en Flex, sigue exactamente la misma estructura, y se sitúa en */control/Controller.js* (*/src/control/Controller.as* en Flex), añadimos las siguientes líneas al final:

```
// Evaluation module commands
this.addCommand(EvaluationEvent.VIEW_PENDING_ASSESMENTS,
                ViewWaitingAssesmentsCommand);
```

```
this.addCommand(EvaluationEvent.VIEW_CURRENTLY_ASSESSED_TO_USER,
                ViewCurrentlyAssesedToUserCommand);
this.addCommand(EvaluationEvent.VIEW_CURRENTLY_ASSESSED_BY_USER,
                ViewCurrentlyAssesedByUserCommand);
```

El segundo parámetro del método `addCommand` es el comando que se llamará cuando se capture el evento mencionado en el primer parámetro del mismo método.

B.3.2. Creación de los comandos

Los comandos se encuentran ordenados por carpetas de cada módulo dentro de */commands/*, al igual que en Flex dentro de */src/commands/*.

Los comandos extienden de `Cairngorm.Command`, y todos implementan la función `execute`. Además, aquellos comandos encargados de cargar contenido, también tienen dos métodos llamados `onResult` y `onFault` que actuarán cuando se reciba el contenido de forma dinámica. En resumen, todos los comandos que piden contenido tienen una estructura de este estilo (para el caso de WaitingAssesments):

```
var ViewWaitingAssesmentsCommand = Cairngorm.Command.extend(
{
  execute : function ()
  {
    var _this = this;

    BP.CMS.prepareMainContent("pending assesments", function ()
    {
P.EvaluationDelegate.viewPendingAssesments(_this);
    });
  },

  onResult : function ( response )
  {
    BP.SM.pushState("Evaluate :: Pending Assesments - Babelium Project",
        {module : "evaluate", action : "pending"});
    BP.CMS.innerMainContent(response);
  },

  onFault : function ()
  {
    BP.CMS.abortLoading();
    alert("Error loading pending assesments");
  }
});
```

BP.CMS (o *Content Management System*) es una de las clases *helper* de la librería

babelium.js. Contiene funciones para desplegar, preparar u ocultar contenido en la página web.

BP.CMS.prepareMainContent elimina el módulo/sección que está cargado en estos momentos y muestra la ventana de loading. Recibe dos parámetros, el primero un string para mostrar mientras se carga, el segundo una función callback que se llamará cuando termine.

BP.EvaluationDelegate es el delegado encargado de pedir los contenidos del módulo de evaluación. En este caso hemos inventado la función que necesitamos, viewPendingAssesments, y como todos los métodos de los delegados, necesitan obligatoriamente un parámetro identificando a la clase responder, es decir, aquella con métodos onResult y onFault para actuar cuando el delegado recupere la información (que en este caso, es el mismo comando el que actúa como responder).

BP.SM.pushState , `BP.SM` es el StatusManager de babelium.js, se encarga de gestionar el estado de la aplicación para permitir carga dinámica y estática mediante la API history de HMTL5 (solamente si el navegador la soporta). El método `pushState` recibe dos parámetros, título de la sección que se va a cargar, y los parámetros que serán traducidos a estado + URL, es decir:

- Ej: {module : "evaluate", action : "pending"} será posteriormente traducido en la URL a *?module=evaluate&action=pending*

BP.CMS.innerMainContent , una vez el gateway ha devuelto la información, se envía directamente al `BP.CMS` que la parsea y la inserta en la web.

NOTA: No se detallará la creación de los otros dos comandos, pues la estructura es prácticamente idéntica; utilizan la misma plantilla.

B.3.3. Adición de los métodos al delegado

El delegado se encuentra en */business/*, al igual que en Flex en la carpeta */src/business/*, y ya están todos los delegados creados, así que sólo tenemos que añadir los métodos que acabamos de crear. Por ejemplo, para el caso de WaitingAssesments, abrir el **EvaluationDelegate** y añadir:

```
1 viewPendingAssesments : function ( responder )
2 {
3   var _service = Cairngorm.ServiceLocator.getHttpService(_serviceID);
4   _service.call( {action : "pending"}, responder );
5 },
```

1. La librería Cairngorm contiene un service locator, que contiene objetos de llamadas HTTP ya programados. `_serviceID` es una variable interna del delegado, y dicha línea obtiene el servicio Cairngorm.*HTTPService* asociado al módulo Evaluate.

2. Los *HTTPService* contienen un método `call`, que reciben los datos que se enviarán por POST. En estas acciones de cambio de vista simples, no hay parámetros complejos, así que sólo contamos con el nombre de la acción a realizar: "pending".

B.3.4. Adición de las clases js creadas y compilación de babelium.js

La última tarea es simple, todo lo que acabamos de crear, hay que añadirlo al **PATH** de compilación de la librería babelium.js. Eso se hace en */build.xml*:

```
<arg value="--js"/><arg value="events/EvaluationEvent.js"/>
<arg value="--js"/><arg value="commands/evaluation/
                                ViewWaitingAssesmentsCommand.js"/>
<arg value="--js"/><arg value="commands/evaluation/
                                ViewCurrentlyAssessedToUserCommand.js"/>
<arg value="--js"/><arg value="commands/evaluation/
                                ViewCurrentlyAssessedByUserCommand.js"/>
```

Y para terminar, compilamos la librería babelium.js, desde eclipse, o desde la consola:

```
ant LH_babeliumjs
```

El siguiente paso, será añadir a la lógica de carga del servidor las 3 acciones que acabamos de crear, para que devuelvan el contenido correspondiente.

B.4. Lógica de representación de la acción número 1

En este caso, asociamos la primera acción con el contenido de la primera de las acciones, que además, es la acción por defecto cuando carga la página. Cómo las 3 acciones son similares, se detallará como insertar esta acción o estado en el módulo evaluation, y obviarán las otras dos.

Para ello vamos a acceder una vez más a **MEvaluation.php** (sección B.1), que se encarga de cargar el módulo, pero primero, vamos a analizar un poco la versión Flex para ver cómo funciona (accediendo al fichero principal del módulo):

```
1 EvaluationMain.mxml@172:
2 evaluationOptionsViewStack.addChild(waitingAssessmentBoxNavContent);
```

Es la primera pestaña que carga, así que también serán nuestra primera pestaña y/o acción.

```
1 EvaluationMain.mxml@281:
2 EvaluationEvent(EvaluationEvent.GET_RESPONSES_WAITING_ASSESSMENT).
      dispatch();
```

Vemos que carga los assesments a través de ese evento. Si seguimos la lógica cairngorm, del evento EvaluationEvent hasta el delegado, vemos que llama al siguiente método de la API:

```
1 EvaluationDelegate.mxml@24:
2 var pendingCall:AsyncToken=service.getResponsesWaitingAssessment();
```

Así que vamos a hacer exactamente lo mismo, abrimos **MEvaluation.php** y añadimos al final:

```
1 if ( !isset(self::$action) || self::$action == "pending" )
2 {
3   $evaluation = new Evaluation();
4   $response = $evaluation->getResponsesWaitingAssessment();
5
6   $content .= WidgetLoader::loadWidget("EvaluationDataTable", $response, self::$actio
7 }
```

Si no se ha especificado ninguna acción, o la acción en cuestión es "pending" (haciendo referencia a *waiting for assesments*), se accede a la clase `Evaluation` de la API y se obtienen las respuestas pendientes de evaluación (variable `$response`). Acto seguido, se carga el widget genérico que muestra la lista de ejercicios en el módulo evaluation, recibiendo como parámetros la lista de respuestas y el estado en el que se encuentra el módulo.

B.4.1. Widget de genérico de carga de evaluaciones

Acto seguido, procederemos a crear el widget de `EvaluationDataTable` que acabamos de pedir que cargue el *WidgetLoader* (y asumiremos que se ha añadido a la base de datos de *WidgetLoader* para que conozca su existencia):

```
1  <?php
2  require_once(dirname(__FILE__) . "/../Widget.php");
3
4  // Utils
5  require_once(dirname(__FILE__) . "/../../util/view/LevelCorrespondence.php");
6  require_once(dirname(__FILE__) . "/../../util/view/LocaleFlagResource.php");
7
8  class WEvaluationDataTable extends Widget
9  {
10   public static function load($args)
11   {
12     parent::load($args);
13
14     // Prepare template
15     self::assign("data", $args[1]);
16     self::assign("action", $args[2]);
17     self::assign("cfg", self::$config);
18     self::assign("locale", new LocaleFlagResource());
19     self::assign("level", new LevelCorrespondence());
20
21     return self::fetch("evaluation/EvaluationDataTable.tpl");
22   }
23 }
24 ?>
```

En este caso nos encontramos con un widget un poco más complejo, que carga una plantilla **.tpl**, como todos, pero que requiere de más parámetros para que Smarty pueda procesarlo.

data hace referencia al primer argumento que recibe el widget, que siguiendo el código del módulo creado arriba, hace referencia a la lista de ejercicios, un array donde cada elemento contiene información de un ejercicio a realizar.

action es el estado en el que se encuentra el módulo (waiting, touser o byuser) en función de que pestaña haya elegido ver

cfg es la configuración de la aplicación, que Smarty usará para calcular rutas a imágenes

locale : esta, y la siguiente, son clases que hacen de *helpers* para traducir información codificada de la base de datos en información visual y útil. Por ejemplo, locale es una instancia de la clase *LocaleFlagResource* que traduce un idioma de vídeo (por

ejemplo "eu_ES") en un icono en forma de bandera (formato HTML) para la web (por ejemplo, la ikurriña).

level : de forma similar al anterior, este traduce niveles de conocimiento del 1 al 5 alojados en la base de datos en su correspondiente nivel de comprensión de un idioma (A1, A2, B1, B2 y C1).

B.4.2. Representación de la acción número 1

Vamos a la carpeta de las plantillas (recordamos */themes/babelium/templates*) y creamos evaluation/EvaluationDataTable.tpl, que de acuerdo a lo comentado anteriormente, quedaría algo así:

```
ection class="VBox">
<header>
  <h1>
f empty($action) || $action == "pending"}
  {assign var="itemRender" value="/evaluation/WaitingItemRender.tpl"}
    {i18n name="EPAITUGABEDAUDENAK"}
lseif $action == "byuser"}
  {assign var="itemRender" value="/evaluation/ByUserItemRender.tpl"}
    {i18n name="NIKEPAITUTAKOAK"}
lseif $action == "touser"}
  {assign var="itemRender" value="/evaluation/ToUserItemRender.tpl"}
    {i18n name="NIRIPAITUTAKOAK"}
if}
  </h1>
</header>

nclude file="/util/Pagination.tpl"}

<div class="assesmentsContainer">
oreach from=$data item=evaluation name=evaluations}
  {include file=$itemRender}
foreach}
</div>

nclude file="/util/Pagination.tpl"}

section>
```

Resumiendo, primero podemos observar que hay un `header` (cabecera) donde, en función del estado del módulo se cargará un título distinto y un *itemRender* distinto. Supongamos para este caso (el que se está explicando a modo de ejemplo) que el módulo está en estado WaitingAssesments, se cargaría el título "Waiting for Assesments" (asumimos que el plugin i18n devuelve dicha traducción) con el item render `WaitingItemRender`.

Acto seguido, se cargaría el widget de paginación, para los ejercicios que se insertarán a continuación.

Después encontramos un contenedor, y dentro de él, mediante Smarty, un `foreach` que se recorrerá todos los ejercicios pendientes de evaluación del parámetro `$data` (los resultados de buscar en la API), y por cada uno de los resultados, lo mostrará utilizando el *itemRender* mencionado arriba.

ItemRender

Ahora que tenemos la visualización genérica de vídeos, sólo falta crear el *itemRender* que hemos mencionado (WaitingItemRender.tpl) para que cada uno de los vídeos se visualice correctamente. He aquí el código de la plantilla:

```
<article class="evaluation" data-responseId="{$evaluation->responseId}">
  <header>
    <h1 class="evaluationTitle HBox vcenter">
     <img src="{$locale->getFlagResource($evaluation->exerciseLanguage)}"
            width="16" height="16" alt="flag"/>
     <p class="level">{$level->getLevelCorrespondence(
            $evaluation->exerciseAvgDifficulty)}</p>
     <p>{$evaluation->exerciseTitle}</p>
    </h1>
  </header>

  <div class="evaluationItemContainer">
   <img src="{$cfg->imagePath}/{$evaluation->exerciseName}/default.jpg"
          alt="{$evaluation->exerciseTitle}"
          width="120" height="90" align="left"/>
   <img src="{$cfg->imagePath}/{$evaluation->responseFileIdentifier}/default.jpg"
          alt="{$evaluation->exerciseTitle}"
          width="120" height="90" align="left" />

    <div>
     <p class="username">
        Recorded by: {$evaluation->responseUserName}
     </p>

     <p class="recordDate">
        Record date: {$evaluation->responseAddingDate}
     </p>

     <p>
        Assesments count: {$evaluation->responseRatingAmount}
     </p>

     <p>
        Role: {$evaluation->responseCharacterName}
     </p>
    </div>
```

```
37   </div>
38 </article>
```

A modo de aclaración, el objeto `$evaluation` dentro de Smarty, hace referencia a cada uno de los ejercicios pendientes de evaluación (ver `foreach` de la sección anterior), es por ello que Smarty utiliza dicho objeto para obtener la información en su interior. Además, se puede observar cómo utiliza `$locale` y `$level` para obtener la ruta de la imagen de la bandera del idioma del vídeo, y el nivel del conocimiento del vídeo correspondientemente, apartir de únicamente el objeto `$evaluation`.

Repitiendo los pasos anteriores, se puede conseguir un módulo con estados y subestados tan complejo como se desee. El módulo Evaluation es más extenso y complejo, pero conllevaría una redacción de otra memoria solamente detallar su completo funcionamiento. Por lo que se presupondrá que lo explicado hasta el momento es más que suficiente para entender lo que falta.

Apéndice C

Comparación de eficiencia entre Flex y HTML5

Contents

En este capítulo se mostrarán comparaciones entre las 2 versiones de Babelium Project (la de Flex y la que contempla esta memoria, en HTML5) en 3 puntos distintos: tiempo de carga y tasa de transferencia en la primera carga; tiempo de carga y tasa de transferencia en la navegación entre secciones; y, uso de CPU y memoria en la navegación por la web.

C.1. Comparación en la primera carga

La primera vez que se cargan las páginas, la versión de Flex necesita descargar el SWF (programa Flash compilado) completo, mientras que la versión HTML5 sólo necesita descargar las librerías JavaScript y los estilos CSS necesarios, así como la estructura HTML.

Para esta prueba se ha utilizado la consola de Chrome para calcular los bytes transferidos y la velocidad de carga de la página. En las siguientes figuras puede verse la comparación.

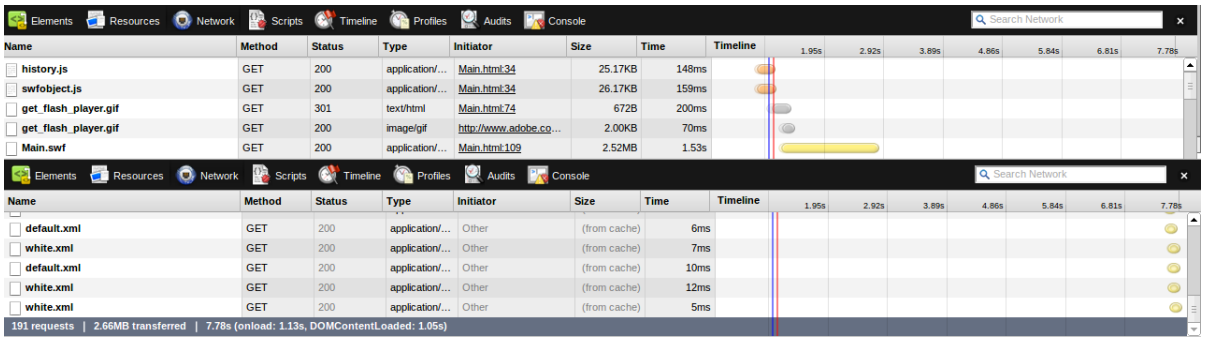


Figura C.1: Eficiencia (Flex): Carga inicial

La versión Flex transfiere **un total de 2,66MB** (se puede observar en la parte de abajo de la figura C.1) de los cuales 2,52MB representan el archivo SWF. La carga completa de la web se realiza **en 7,78 segundos**. Además, dado que es Flex el que se encarga de recibir y cargar todas las imágenes de los vídeos, dichas transferencias no aparecen en la tasa de bytes transferidos, por lo que podría ser aún mayor.

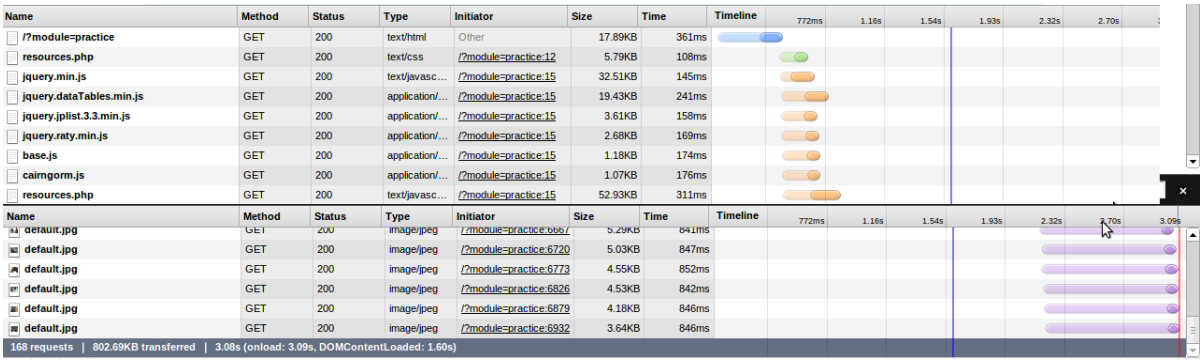


Figura C.2: Eficiencia (HTML5): Carga inicial

La versión HTML5, en comparación, transfiere aproximadamente 132KB que se refieren al contenido de la web (?module=practice) y a los scripts JS y CSS necesarios para visualizarla. Luego está el resto de la carga relativa a la transferencia de imágenes (que en la figura de Flex no se refleja) hasta **un total de 802KB** (que sigue siendo bastante menor que en la versión de Flex) **en 3,08 segundos**.

C.2. Comparación en la carga de contenidos

En esta sección se muestra la diferencia entre la cantidad de bytes transmitidos al llamar a los servicios para obtener la lista de ejercicios disponibles. Mientras que la versión de Flex utiliza el protoclo AMF encapsulado en la respuesta HTTP para devolver datos, la versión HTML5 devuelve una string de datos JSON en el mismo cuerpo HTML.

Para esta prueba se ha utilizado Charles Proxy[1].

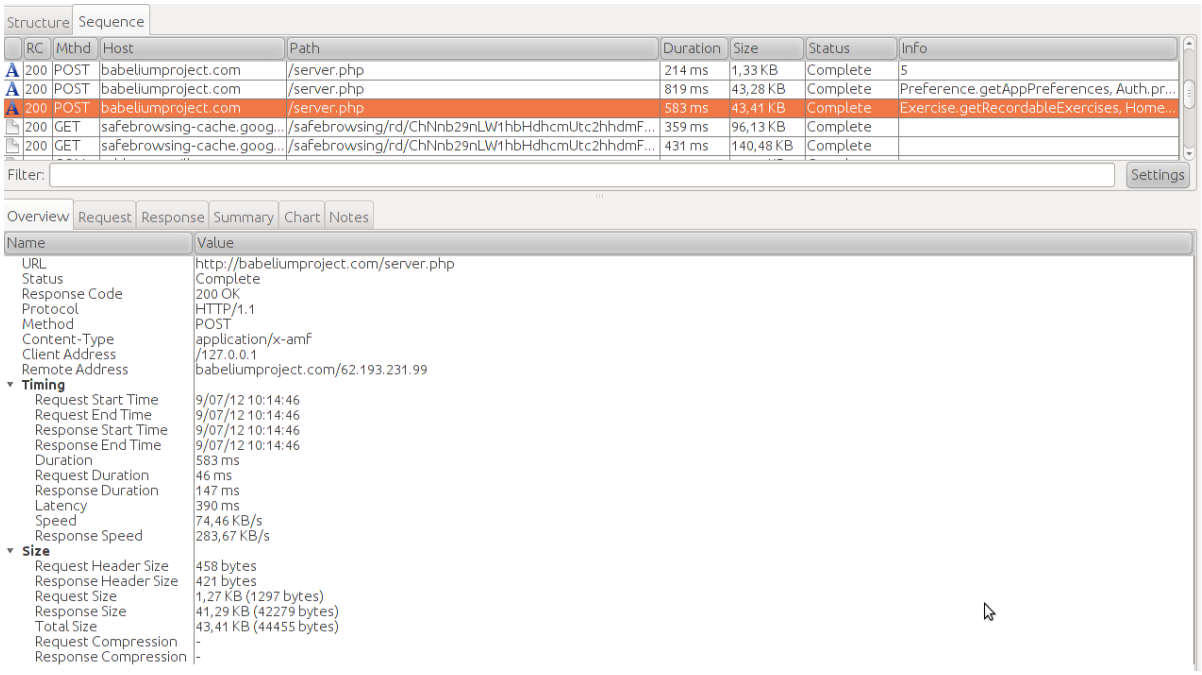


Figura C.3: Eficiencia (Flex): Obtener ejercicios

Pese a que Flex realiza más de una llamada a los servicios al mismo tiempo y en la respuesta están encapsuladas respuestas a 4 llamadas, el 90 % del contenido es referente al listado de ejercicios, cuya transferencia ocupa un **total de 40KB aproximadamente** y se realiza **en 550ms**.

Por el contrario, HTML5 tan sólo transmite un **total de 17,47KB en 368ms**, dato curioso, pues HTML5, aún transmitiendo etiquetas y maquetación en la respuesta, transmite menos bytes que Flex, que sólo envía el objeto con los datos, sin enviar etiqueta alguna. Esto se debe a que la respuesta de Flex va encapsulada bajo el protocolo AMF

[1]Plugin utilizado para hacer debug HTTP en la versión de Flex ya que soporta protocolos AMF

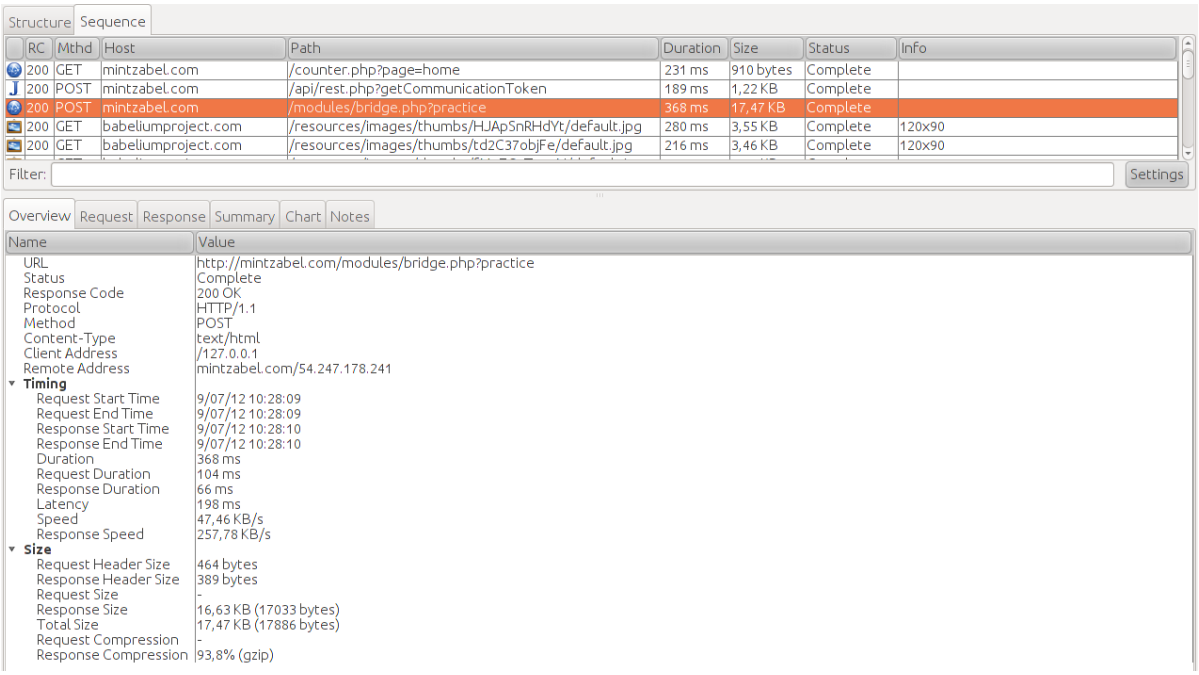


Figura C.4: Eficiencia (HTML5): Obtener ejercicios

dentro de HTTP y dicho protocolo añade muchos datos de control para que el objeto pueda ser recibido en el cliente.

Las figuras C.5 y C.6 muestran la respuesta RAW de Flex y HTML5 respectivamente.

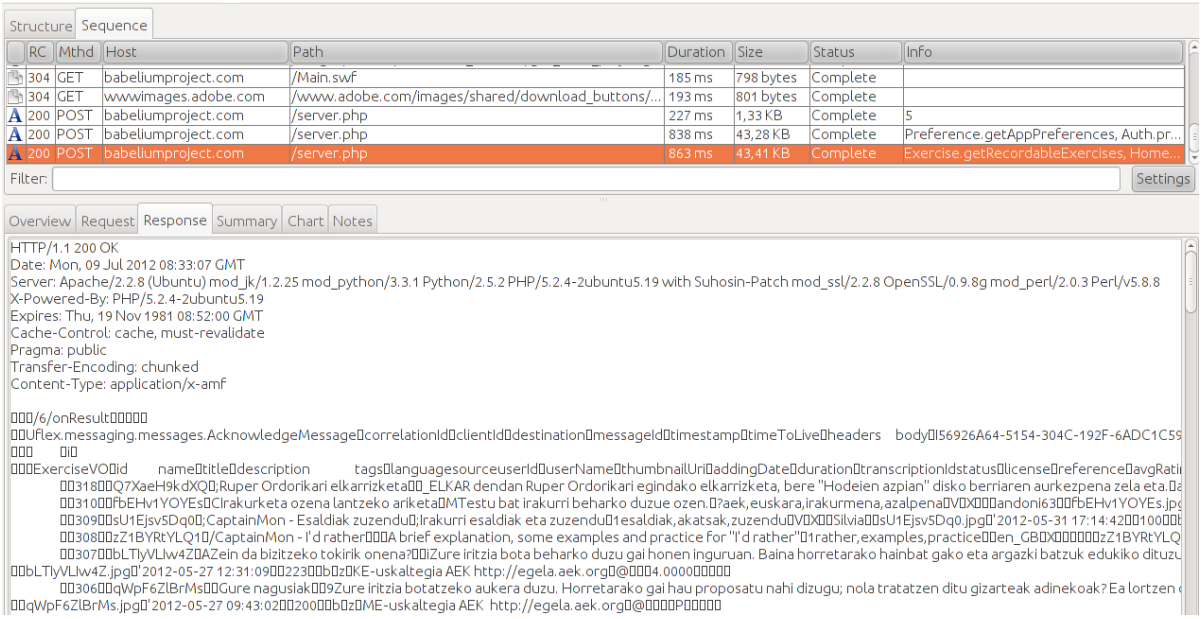


Figura C.5: Eficiencia (Flex): Respuesta de ejercicios

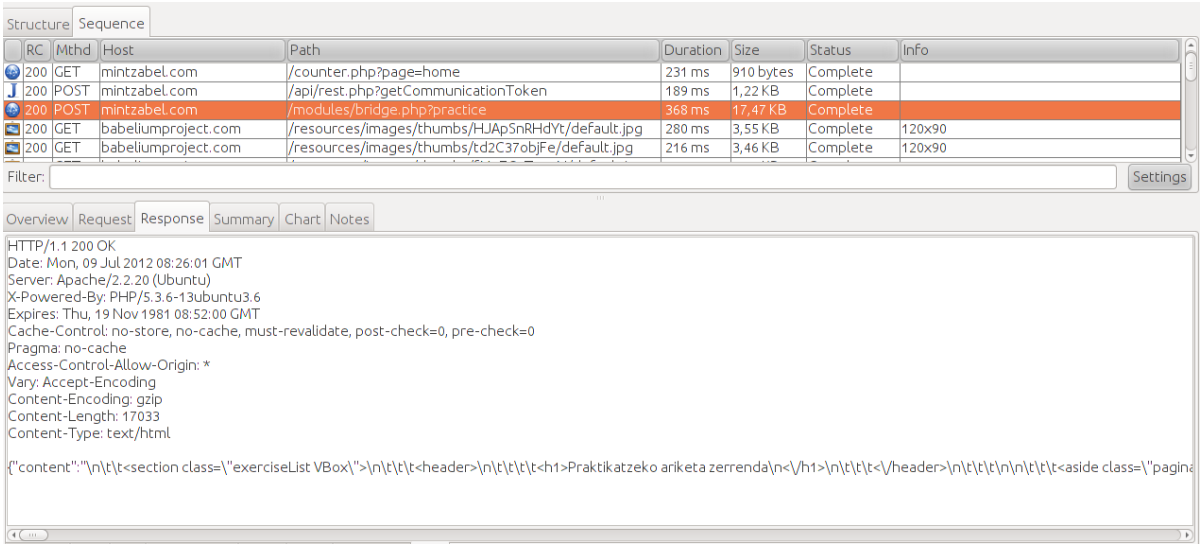


Figura C.6: Eficiencia (HTML5): Respuesta de ejercicios

C.3. Comparación en el uso de Memoria y CPU

Para esta prueba se ha realizado el Administrador de tareas de Chrome, teniendo únicamente una pestaña abierta con la web en cuestión, y se comparan la memoria y uso de CPU en 3 casos: carga inicial, navegación entre secciones y reproducción de un vídeo.

En la carga inicial, una vez cargada la página, la pestaña de la versión Flex consume menos memoria, sin embargo, el plugin de Flash está consumiendo 157MB de memoria mientras que en la versión HTML5 tan solo 65MB, que es lo que consume en *standby* (sólo por estar cargado y no estar en uso). En ambas versiones, el plugin de Flash en el caso de Flex y la pestaña de la web en el caso de HTML5, oscilan el uso de CPU entre 3 y 20 % aproximadamente, estabilizándose en 0 % una vez cargada la página.

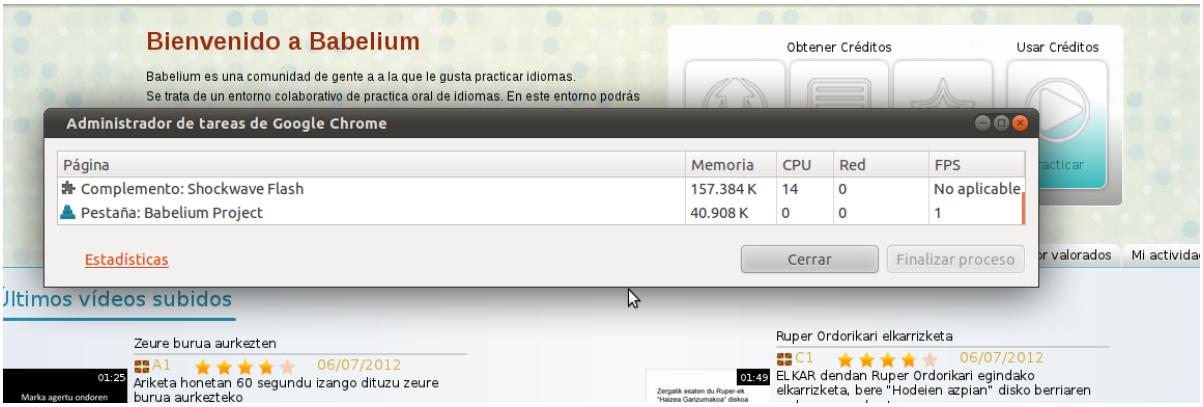


Figura C.7: Eficiencia (Flex): Consumo de recursos inicial

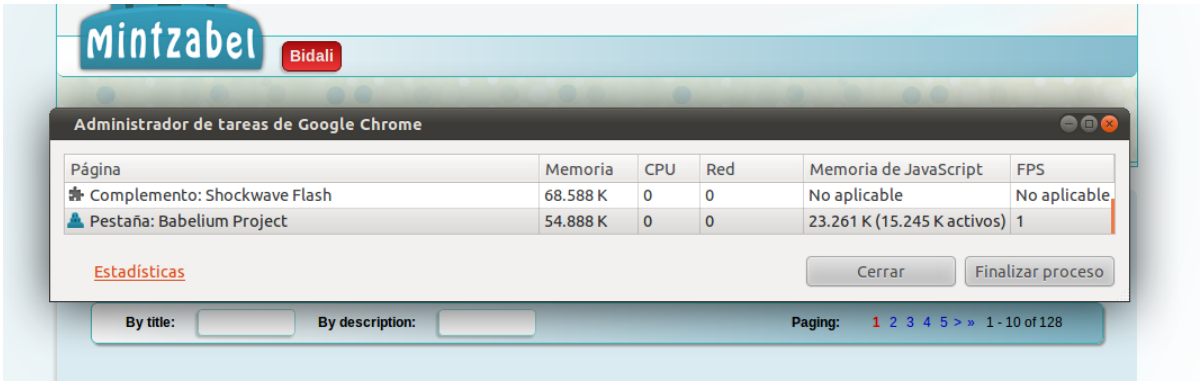


Figura C.8: Eficiencia (HTML5): Consumo de recursos inicial

Al cargar la sección practice, la versión de Flex aumenta el uso de memoria (en el plugin Flash) hasta 170MB, mientras que la versión HTML5 aumenta a 61MB de memoria utilizados por la pestaña de la web. Así mismo, la versión HTML5 hace uso de efectos para ocultar y añadir nuevo contenido, que se reproducen a 20FPS. Ambas versiones oscilan el uso de CPU entre 5 y 20 % aproximadamente.

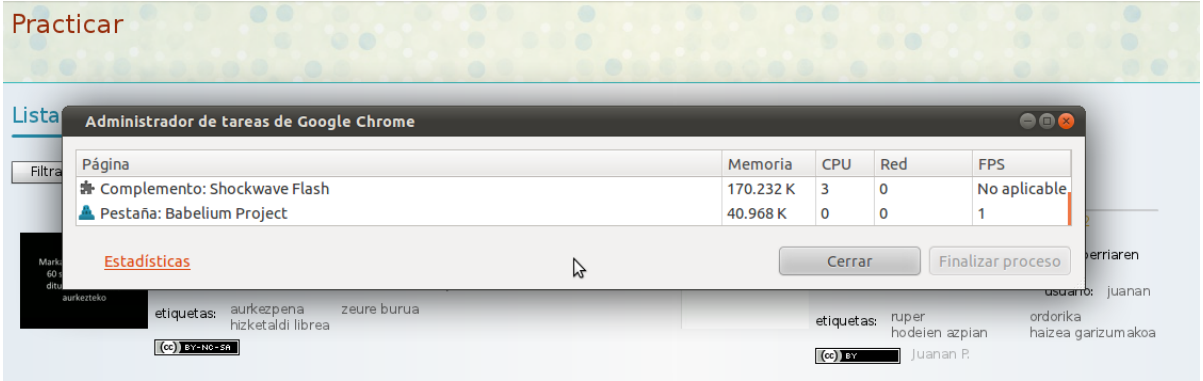


Figura C.9: Eficiencia (Flex): Consumo de recursos al cambiar de sección

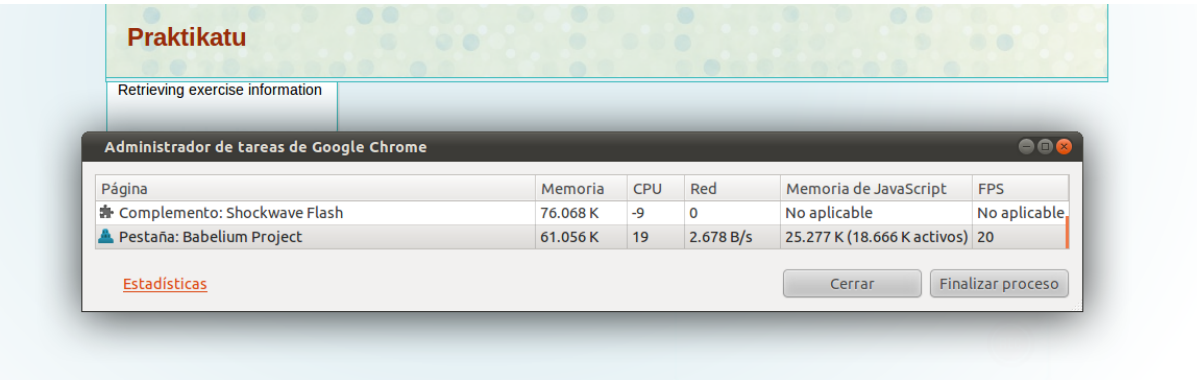


Figura C.10: Eficiencia (HTML5): Consumo de recursos al cambiar de sección

Por último, mientras se reproduce un ejercicio, la memoria utilizada del plugin de Flash en la versión Flex asciende hasta 180MB mientras que en la versión HTML5 sigue estable en 76MB, y el uso de CPU del plugin de Flash en ambas versiones oscila entre los 10 y los 30 % (con valores más altos en la versión de Flex).

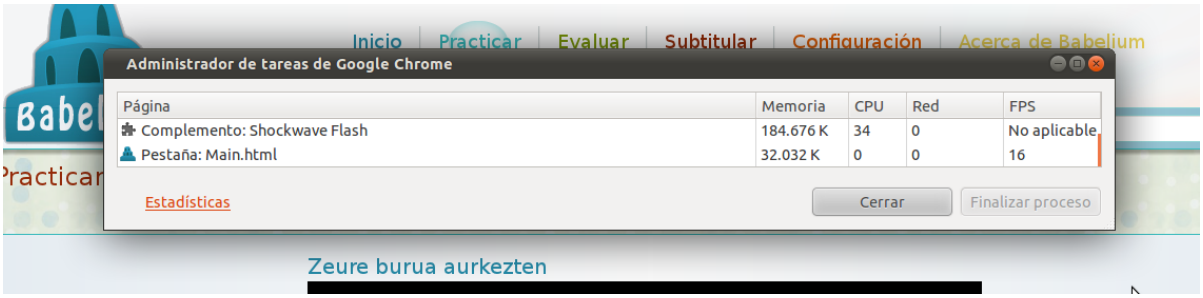


Figura C.11: Eficiencia (Flex): Consumo de recursos al reproducir un ejercicio

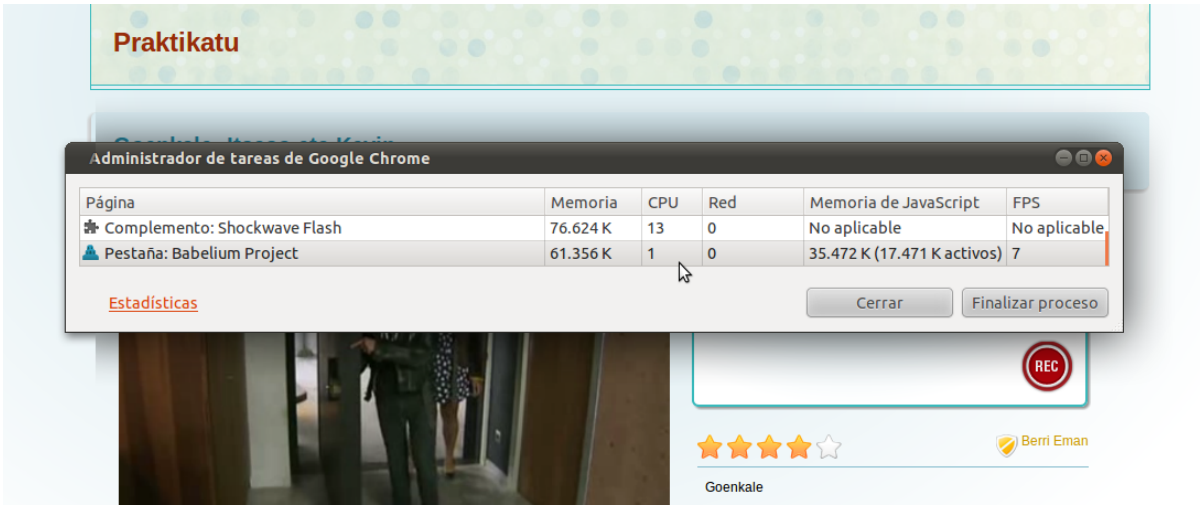


Figura C.12: Eficiencia (HTML5): Consumo de recursos al reproducir un ejercicio

C.4. Resumen

Los bytes y el tiempo hacen referencia a la tasa y duración de la recepción de información:

	Bytes	Tiempo	Memoria	CPU
Carga inicial				
Flex	2,66MB	7,78s	Flash: 157MB; Pestaña: 40MB	3-20 %, 0 %
HTML5	802KB	3,08s	Flash: 68MB; Pestaña: 54MB	3-20 %, 0 %
Carga de una sección				
Flex	40KB	550ms	Flash: 170MB; Pestaña: 40MB	3-15 %
HTML5	17,4KB	368ms	Flash: 76MB; Pestaña: 61MB	3-20 %
Reproducción de un ejercicio				
Flex	-	-	Flash: 184MB; Pestaña: 32MB	15-40 %
HTML5	-	-	Flash: 76MB; Pestaña: 61MB	5-20 %

Cuadro C.1: Comparativa de recursos utilizados por Flex y HTML5

Apéndice D

Sobre Babelium Project

Contents

La siguiente sección está sacada del artículo [31], modificada ligeramente.

D.1. [PR] BabeliumProject.com: practica tu Euskera, mejora tu Inglés

BabeliumProject.com es una aplicación web para la práctica oral y colaborativa de idiomas desarrollada por el grupo de investigación GHyM de la Universidad del País Vasco / Euskal Herriko Unibertsitatea. El usuario puede usar Babelium directamente desde su navegador sin necesidad de instalar ninguna otra aplicación.

Babelium Project permite al usuario visualizar ejercicios en distintos idiomas. Estos ejercicios son grabaciones de vídeo que permiten al usuario practicar idiomas mediante:

Doblaje de un rol : Existen vídeos de conversaciones donde intervienen, por lo general, 2-3 personas. El usuario puede tomar el papel de uno de los personajes y realizar el doblaje del mismo. Es decir, una vez seleccionado el rol que quiere doblar, el vídeo comienza a visualizarse hasta que llega el turno de dicho rol; en ese momento se graba la voz del usuario (a través del micrófono y, si el usuario lo desea, también la imagen de su webcam). Tal y como ocurre en la vida real, el usuario dispone de un tiempo limitado para responder, exactamente el mismo que el tiempo utilizado por el rol del personaje en el vídeo original.

Lectura de un texto : También hay disponibles vídeos con relatos o textos escritos, donde el usuario deberá grabarse leyendo el texto que aparece en el vídeo para comprobar su nivel de lectura en voz alta.

Práctica oral libre : De una forma similar a como funcionan las pruebas orales en los exámenes de idiomas, también hay disponibles vídeos donde van apareciendo imágenes sobre un tema (por ejemplo, el calentamiento global), y se le da al usuario un tiempo para que hable libremente sobre ese tema.

Cuando el usuario termina la realización del ejercicio, tiene la opción de visualizar su trabajo. En cuanto determina que la grabación es correcta, la puede subir al servidor Babelium de tal forma que queda disponible para su evaluación por parte de otros usuarios. Esta evaluación se realiza de forma colaborativa, es decir, si la grabación a evaluar se ha hecho en euskera, serán usuarios euskaldunes los que evalúen el trabajo. Esta evaluación puede incluir tanto comentarios de texto como comentarios en vídeo, actuando como profesores de ese idioma.

Las video-conversaciones disponibles en Babelium abarcan distintos niveles de dificultad, siguiendo la nomenclatura del Marco Europeo de Referencia de las Lenguas (A1, A2, B1, B2, C1), utilizada en varios organismos oficiales, entre ellos las Escuelas Oficiales de Idiomas. Por otra parte, aunque Babelium dispone ya de vídeos en euskera e inglés, cualquier usuario puede subir sus propias creaciones, en el idioma que desee.

Por otro lado, aparte de servir como herramienta para la práctica oral de idiomas, el usuario puede usarla también para mejorar sus capacidades de comprensión auditiva del idioma, mediante el uso del apartado de subtitulado. Cuando un nuevo vídeo hace su aparición en Babelium, no contiene subtítulos y son los propios usuarios quienes pueden, nuevamente de forma colaborativa, subtitularlo a través de un editor creado para esta labor que está totalmente integrado en Babelium.

Para fomentar la participación y colaboración en el sistema, se introduce el concepto de créditos. Conseguir créditos es una acción gratuita (económicamente hablando), pero requiere que el usuario aporte recursos a Babelium, bien mediante el subtitulado de nuevos vídeos, evaluando el trabajo y grabaciones de otros usuarios o subiendo nuevas video-conversaciones. Por otro lado, si un usuario quiere que otros evalúen su trabajo, tiene que consumir parte de sus créditos. De esta forma, se busca que exista un equilibrio entre lo que se aporta y lo que se recibe del sistema, al mismo tiempo que se evitan posibles abusos.

Cabe destacar que Babelium se ha lanzado como una aplicación gratuita, con su código ya liberado. Esto implica, que los usuarios con más conocimientos técnicos tendrán además la posibilidad de estudiar, analizar y mejorar el código del producto, con el objetivo de que Babelium se convierta en una mejor herramienta para la práctica de idiomas.

Los responsables de BabeliumProject informan de novedades y responden a preguntas y cuestiones a través del usuario @babelium en Twitter.

Este proyecto ha sido posible gracias a la colaboración de HABE, la Diputación Foral de Gipuzkoa, y el Gobierno Vasco.

Bibliografía

[1] Blogs, Adobe Featured: *Adobe abandona Flash para dispositivos móviles para centrarse en HTML5.* http://blogs.adobe.com/conversations/2011/11/flash-focus.html.

[2] Paul, Ryan: *Adobe donates Flex to foundation in community-friendly exit strategy.* http://arstechnica.com/business/2011/11/adobe-donates-flex-to-foundation-in-community-friendly-exit-strategy/.

[3] Muntión, Imanol Luengo: *Refactorización, mejora y adición de nuevas funcionalidades para Babelium Project*. 2010.

[4] Corporation, Kaazing: *HTML5 Overview.* http://kaazing.com/training/html5-overview.

[5] *HTML5 Rocks.* http://html5rocks.com/.

[6] Network, Mozilla Developer: *Mozilla HTML5 Demos.* https://developer.mozilla.org/es/demossearch?q=html5.

[7] Juan A. Pereira, Silvia Sanz, Inko Perurena Julián Gutiérrez y Luengo, Imanol: *An experience migrating a Cairngorm based Rich Internet Application from Flex to HTML5.* Jornadas de Ingeniería Software y Bases de Datos 2012 (JISBD 2012), 2012.

[8] Software, Fog Creek: *Trello.* https://trello.com/.

[9] Luengo, Imanol: *Migración FLEX a HTML5.* http://html5.babeliumproject.com/.

[10] Adobe, Open @: *Overview of the Cairngorm Architecture.* http://sourceforge.net/adobe/cairngorm/wiki/UnderstandingCairngorm/.

[11] EllisLab, Inc: *CSS3.info.* http://www.css3.info/preview/.

[12] Foundation, The jQuery: *JQuery: The Write Less, Do More, Javascript Library.* http://jquery.com/.

[13] Labs, Adobe: *Wallaby: Convert Adobe Flash FLA files into HTML and reach more devices.* http://labs.adobe.com/technologies/wallaby/.

[14] jangaroo.net: *Jangaroo: AS3 without Flash Plugin.* http://www.jangaroo.net/home/.

[15] Labs, Adobe: *FalconJS: an ActionScript to JavaScript cross-compiler based on Falcon.* http://blogs.adobe.com/bparadie/2011/11/19/what-is-falconjs/.

[16] Google: *Google Swiffy.* https://www.google.com/doubleclick/studio/swiffy/.

[17] SourceTec Software Co., LTD: *Sothink SWF to HTML5.* http://www.sothink.com/.

[18] Franck Fleurey, Erwan Breton, Benoit Baudry Alain Nicolas y Jézéquel, Jean Marc: *Model-Driven Engineering for Software Migration in a Large Industrial Context.* 2007.

[19] Adobe: *About Spark layouts.* http://help.adobe.com/en_US/flex/using/WS0141D24B-6AEB-4721-BA04-9BF15F86350F.html.

[20] Adobe: *The Spark BorderContainer container.* http://help.adobe.com/en_US/flex/using/WS03d33b8076db57b9466e6a52123e854e5d5-8000.html.

[21] Russell, Alex: *Flexible Boxes from Flex to HTML5.* http://infrequently.org/2009/08/css-3-progress/.

[22] Canyon, Code: *jQuery jPList Plugin.* http://codecanyon.net/item/jquery-jplist-plugin/1860318.

[23] SpryMedia: *DataTables - Table plugin for JQuery.* http://www.datatables.net/index.

[24] Hempton, Gordon L.: *The Top 10 Javascript MVC Frameworks.* http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/.

[25] Adobe, Open @: *Open Source Cairngorm Framework.* http://sourceforge.net/adobe/cairngorm/home/Home/.

[26] Resig, John: *Simple JavaScript Inheritance.* http://ejohn.org/blog/simple-javascript-inheritance/.

[27] Developers, Google: *Google Closure Compiler.* https://developers.google.com/closure/compiler/.

[28] elated.com: *JavaScript and Cookies*. http://www.elated.com/articles/javascript-and-cookies/.

[29] Millikin, John: *How to create a GUID / UUID in Javascript?* http://stackoverflow.com/questions/105034/how-to-create-a-guid-uuid-in-javascript.

[30] UPV/EHU, GHyM Grupo Hipermedia y Multimedia de: *Babelium / Moodle integration plugin*. http://www.ehu.es/ehusfera/ghym/2011/11/18/babelium-moodle-integration-plugin/.

[31] Juan A. Pereira, Silvia Sanz: *BabeliumProject.com: practica tu Euskera, mejora tu Inglés*.

[32] @rem: *HTML5 Demos and Examples*. http://html5demos.com/.

[33] Nadel, Ben: *JQuery Presentation*. http://www.bennadel.com/resources/ presentations/jquery/video/index.htm.

[34] Grakalic, Alen: *Easy Framework*. http://easyframework.com/documentation.php.

[35] Cake Software Foundation, Inc.: *CakePHP Framework*. http://cakephp.org/.

[36] EllisLab, Inc: *Code Igniter PHP Framework*. http://codeigniter.com/.

[37] Kosyakov, Max: *PHP Template Engines*. http://www.worksforweb.com/publications/php-template-engines/.

[38] Jaspal: *Best HTML5 And CSS3 Frameworks*. http://www.webdesignish.com/best-html5-and-css3-frameworks.html.

[39] Adam Bergkvist, Nicklas Sandgren, Stefan Håkansson Jonas Lundberg y Brodin, Per Erik: *Beyond HTML5 - Conversational voice and video implemented in Webkit GTK+*. https://labs.ericsson.com/developer-community/blog/beyond-html5-conversational-voice-and-video-implemented-webkit-gtk.

[40] Ian: *Create a REST API with PHP*. http://www.gen-x-design.com/archives/create-a-rest-api-with-php/.

[41] Lawson, Bruce: *HTML5: difference between articles and sections*. http://www.brucelawson.co.uk/2010/html5-articles-and-sections-whats-the-difference/.

[42] brunildo.org: *CSS3: Overflow-x, overflow-y tests*. http://www.brunildo.org/test/Overflowxy2.html.

[43] software, Opera: *HTML5 elements, attributes, and APIs support in Opera Presto 2.7.* http://www.opera.com/docs/specs/presto27/html5/.

[44] Ramirez, Enrique: *Coding An HTML 5 Layout From Scratch.* http://coding.smashingmagazine.com/2009/08/04/designing-a-html-5-layout-from-scratch/.

[45] Doctor, HTML5: *HTML5 Doctor: helping you implement HTML5 today.* http://html5doctor.com/.

[46] aNieto2k: *Las principales diferencias entre HTML5 y HTML4.* http://www.anieto2k.com/2007/06/16/las-principales-diferencias-entre-html5-y-html4/.

[47] Loscavio, Frankie: *Fx complete CSS list.* http://www.loscavio.com/downloads/blog/flex3_css_list/flex3_css_list.htm.

[48] Inc, Adobe: *Tour de Flex.* http://www.adobe.com/devnet-apps/flex/tourdeflex/web/.

[49] Maikel: *Crear e implementar el patrón singleton en PHP.* http://www.cristalab.com/tutoriales/crear-e-implementar-el-patron-de-diseno-singleton-en-php-c256l/.

[50] ProyectosAgiles.org: *Qué es Scrum.* http://www.proyectosagiles.org/que-es-scrum.

[51] W3C: *HTML5: Edition for Web Authors.* http://dev.w3.org/html5/spec-author-view/history.html.

[52] Alex@Net: *Model-View-Controller (MVC) with JavaScript.* http://www.alexatnet.com/articles/model-view-controller-mvc-javascript.

[53] Scripts, Movable Type: *JavaScript implementation of SHA-1 Cryptographic Hash Algorithm.* http://www.movable-type.co.uk/scripts/sha1.html.

[54] Bradford, Anselm: *Object-Oriented JavaScript Tip: Implementing The Singleton Pattern.* http://blog.anselmbradford.com/2009/04/21/object-oriented-javascript-tip-the-quintessential-singleton/.