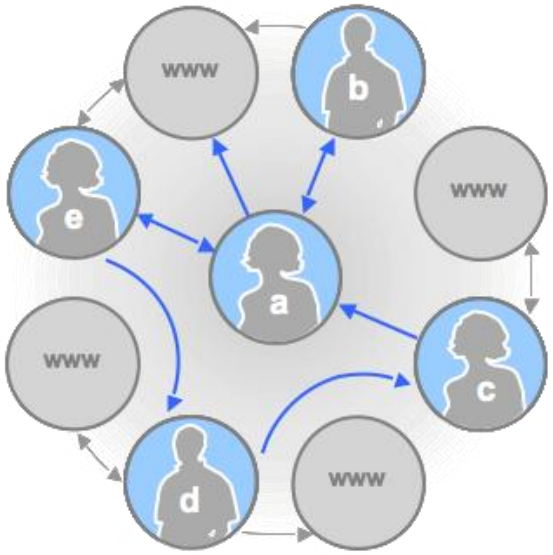
Hacia Tutor 2.0

Aplicando características de la Web 2.0 a Tutor



Alumno: Álvaro López Martínez

Tutor: Miguel J. Hornos Barranco

Curso 2010/2011

Índice

Índice de figuras

Capítulo 1. Introducción

Con el título de "Hacía Tutor 2.0: Aplicando características de la Web 2.0 a Tutor" se define de forma bastante fiel la naturaleza de este proyecto. De este título se hace clave destacar 2 conceptos:

Tutor: Tutor [49] es una plataforma web de apoyo a la docencia universitaria usada y desarrollada principalmente por el departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada. Como plataforma de apoyo a la docencia permite a profesores y alumnos de diversas asignaturas acceder a ficheros, consultar calificaciones, controlar la asistencia de los alumnos… De esta plataforma existen 2 grandes versiones, una primera desarrollada durante los años 2003 a 2006, y una segunda desarrollada desde el 2007 hasta la actualidad. A comienzos del año 2009 se liberó el código fuente de la segunda versión, bajo licencia GNU-GPL3 [12], distribuyéndose a través de la forja de Red Iris [12]. Sobre esta versión se han desarrollado los contenidos de este proyecto.

Web 2.0: El siguiente párrafo, extraído de Wikipedia [53] muestra una definición de Web 2.0:

> **El término Web 2.0 (2004–presente) está comúnmente asociado con un fenómeno social, basado en la interacción que se logra a partir de diferentes aplicaciones web, que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario o D.C.U. y la colaboración en la World Wide Web.(…) Un sitio Web 2.0 permite a sus usuarios interactuar con otros usuarios o cambiar contenido del sitio web, en contraste a sitios web no-interactivos donde los usuarios se limitan a la visualización pasiva de información que se les proporciona.**

Es muy importante tener claro este concepto, pues será citado muy frecuentemente y en torno a él se argumentarán y defenderán las decisiones tomadas y objetivos marcados a lo largo de este proyecto.

Dicho esto, el objetivo principal de este proyecto es sentar las bases para desarrollar una plataforma que permita la colaboración e interoperabilidad entre sus usuarios, y desarrollar diversas funcionalidades en función a estas bases. Para ello será necesario estudiar las características actuales de la plataforma, analizar los requerimientos básicos de una plataforma 2.0, integrar estos en Tutor y desarrollar nuevas funcionalidades.

1.1. Planteamiento y motivación

¿Es útil dotar de funcionalidades de Web 2.0 a una plataforma orientada a la docencia? Esta fue mi primera pregunta a la hora de afrontar este proyecto. Mi conclusión es que si, es útil. Y aunque es una afirmación arriesgada, si bien a día de hoy podríamos considerar esta decisión tan solo de útil, el futuro a corto y medio plazo de Tutor me llevan a pensar que pronto no será solo útil, sino también una necesidad.

Para defender esta argumentación primero hay que hacerse eco del éxito de las denominadas Web 2.0 en la actualidad. Si consultamos el ranking mundial de las webs con más tráfico según Alexa [2]:

Figura 1 - Top webs con más tráficos según Alexa (Fuente: [2])

El ranking es bastante clarificador, Facebook [9], la red social por excelencia, ocupa un segundo puesto. Youtube [60], portal de videos donde cada usuario puede subir sus propias creaciones ocupa el tercero. Y tan solo los buscadores como Google [16], Yahoo [59], Live [31] o Baidu [3] pueden hacer frente al éxito de herramientas 2.0 como Wikipedia [37], enciclopedia colaborativa, Blogger [4], portal de blogs, o Twitter [50], pequeña herramienta de mensajería que permite a sus usuarios comunicarse en pequeños 140 caracteres.

Dicho esto, para seguir defendiendo el uso de funcionalidades Web 2.0 en una plataforma orientada a la docencia hay que entender las nuevas formas de comunicarse e interactuar de un alumno actual. Para ello es importante conocer el concepto de nativo digital. Según el artículo Digital Native de la Wikipedia inglesa [36] y de acorde a la definición dada por Marc Prensky [33] en 2001:

> **"Un nativo digital es una persona nacida después la implantación en general de la tecnología digital y que, como resultado, ha estado familiarizada con tecnologías digitales tales como ordenadores, Internet, teléfonos móviles y MP3s durante toda su vida"**

Según el artículo de nativo digital de la Wikipedia española [37] se puede considerar nativo digital a los nacidos desde 1979 en adelante. De acuerdo a esta definición cualquier prácticamente cualquier alumno universitario es un nativo digital, siendo las nuevas generaciones donde más se nota este hecho.

Para entender el caso que nos concierne, es decir, como afecta ser un nativo digital y el uso de la web 2.0, o en concreto las redes sociales, viene bien analizar las estadísticas mostradas en la Figura 2.

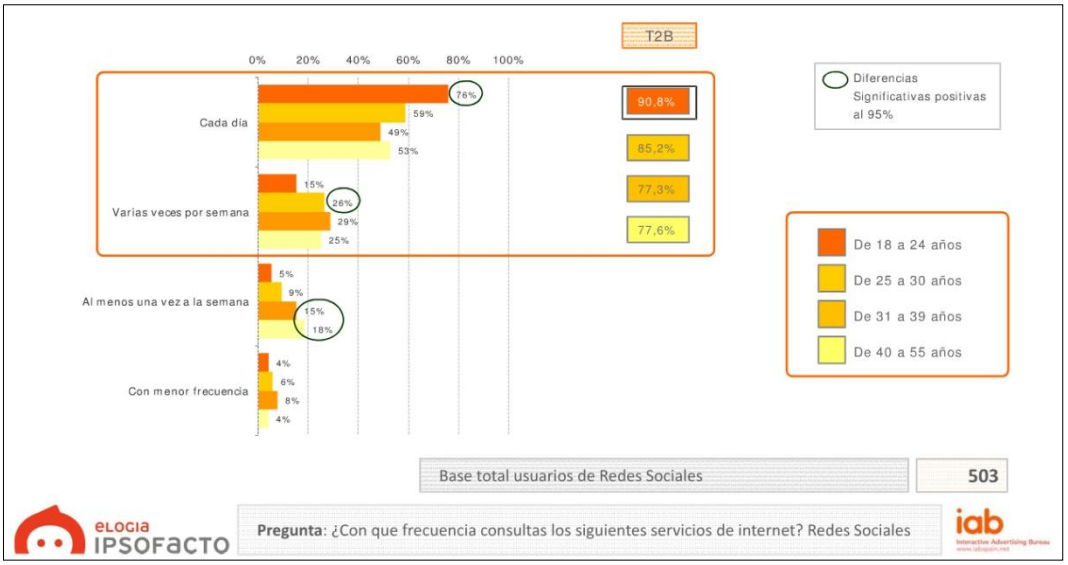


Figura 2 - Uso de las redes sociales según segmentos de edad (Fuente: IAB, Interactive Advertising Bureau)

En esta estadística, obtenida del Interactive Advertising Bureau Spain [23], se aprecia cómo en el segmento de edad de 18 a 24 años un porcentaje del 76% de la muestra consulta redes sociales cada día. Este porcentaje sube hasta un 90,8% si también tenemos en cuenta a quienes visitan la red varias veces por semana. No obstante, la diferencia no es muy notable entre el segmento de edad de 18 a 24 años con respecto al resto de segmentos, pero sí queda claro que la tendencia es que a más joven más frecuencia en el uso de redes sociales.

En el cuadro estadístico de la Figura 3 se aprecia el uso que se hace de las redes sociales, segmentado por la edad de los usuarios:

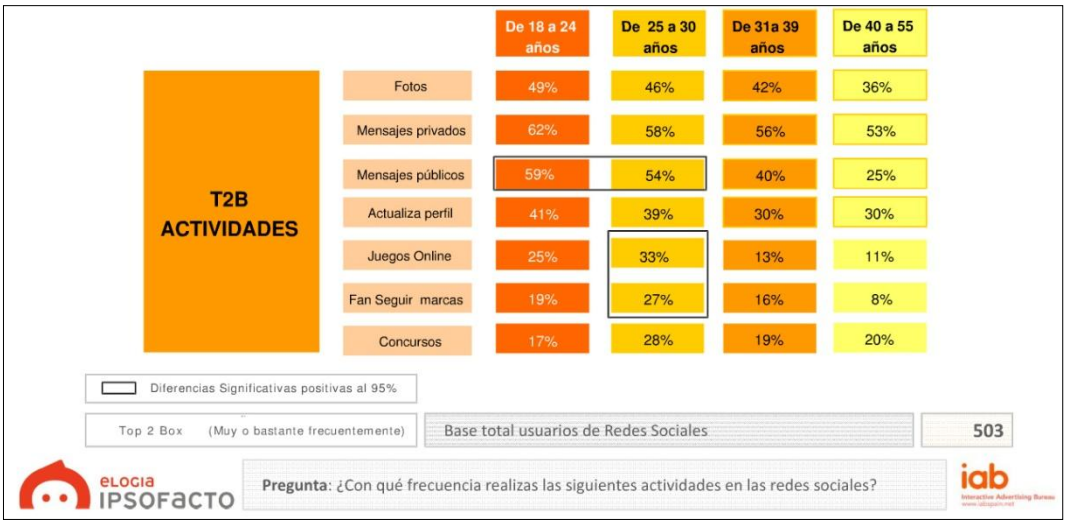


Figura 3 - Actividades en redes sociales (Fuente: IAB, Interactive Advertising Bureau)

Del análisis estadístico de estos datos, el elemento a destacar es que los más jóvenes son los más propensos a hacer uso de los mensajes públicos.

La conclusión a la que se quiere llegar con estos datos y estadísticas de uso es que la forma de comunicarse está cambiando, como ha ido cambiando a lo largo de la historia.

Los nativos digitales se comunican bastante más y de forma más abierta, contando y estando al tanto de lo que hacen sus amistades, vinculando y unificando la vida real con la vida virtual.

Si extrapolamos estos datos a otras herramientas podemos concluir que los nativos digitales ya no sólo reciben contenido, sino que también lo crean, bien mediante blogs, mediante fotos, mediante videos… Internet ha vinculado a otras actividades de tiempo libre, como puede ser la televisión, a un segundo plano, y los nativos digitales han pasado de ser elementos pasivos a elementos totalmente activos, creando sus propios contenidos, comentando los contenidos de otros, valorando ideas,…

Dicho todo esto, la pregunta que hay que formularse es si todos estos hábitos en cuanto a comunicación, colaboración y creación tienen también aplicación en el ámbito de la docencia. Y considero que la respuesta es un sí rotundo. A día de hoy un alumno interesado en un tema puede consultar videos en Youtube, comentar con otros usuarios el video, crear una entrada en su blog, generar un debate mediante su perfil en Facebook, hablar con otros compañeros mediante mensajería instantánea, rebatir una entrada de Wikipedia,…

Por todo ello, y para facilitar a docentes y alumnos ejercer estas nuevas formas de comunicación y aprendizaje, considero útil, e incluso necesario en un futuro, dotar a Tutor de las herramientas que permitan llevar a cabo esta tarea.

1.2. Contenidos de este documento

Esta memoria está dividida en 7 capítulos. El orden en el que están no es casual, responde al hecho de el primer capítulo sienta las bases del resto del trabajo desarrollado. Los capítulos y su contenido son los siguientes:

Capítulo 1: Introducción

Análisis de la motivación de dotar a Tutor de características de la Web 2.0, objetivo de este proyecto.

Capítulo 2: De un diseño monolítico a un diseño modular

Se explican los cambios de diseño adoptados para darle a Tutor características de arquitectura modular. Hasta el momento, estaba desarrollado en una arquitectura monolítica, donde todas las funcionalidades del sistema se concentraban en uno o varios archivos, sin apenas separación a nivel de clases o ficheros.

Capítulo 3: Mensajería interna

Explica cómo se ha llevado a cabo el desarrollo desde cero de un nuevo módulo para Tutor, cuyo objetivo es facilitar la comunicación entre usuarios, más allá de las funcionalidades que pudieran existir en el sistema hasta el momento.

Capítulo 4: Chat

Se analizan diferentes soluciones para la integración de un chat o sistema de mensajería instantánea dentro de la plataforma, realizando la adaptación de una de estas soluciones para que haga uso del sistema de usuarios, grupos y asignaturas de Tutor.

Capítulo 5: Estadísticas y uso del sistema

Desarrollo de una funcionalidad que reutiliza la modularidad implementada en el primer capítulo para generar estadísticas de los diferentes módulos ya existentes en Tutor.

Capítulo 6: Mantenimiento

Detalla las tareas de mantenimiento que requiere un sistema en producción como es Tutor, tareas que, aunque no son exactamente de desarrollo, sí que requieren de una dedicación y estudio profundo del sistema, para mantener requisitos no funcionales, tales como estabilidad, eficiencia, seguridad, consistencia,...

Capítulo 7: Conclusiones y trabajo futuro

Conclusiones sobre el trabajo realizado a lo largo de este proyecto y propuesta de hoja de ruta que debería seguir Tutor a partir de lo desarrollado en el proyecto actual.

Por último, completan esta memoria una sección de anexos y la bibliografía a la que se hace referencia a lo largo de la misma.

Capítulo 2. De un diseño monolítico a un diseño modular

2.1. Motivación

Entender los objetivos propuestos en este capítulo implica conocer cómo estaba construido Tutor antes de aplicar los cambios que el capítulo describe y contextualizar la situación del sistema en el momento de empezar a aplicar los cambios.

El diseño existente antes del desarrollo de este proyecto es herencia de la primera versión de Tutor [52], que estuvo en funcionamiento desde el año 2003 al año 2007. A pesar de que en 2006 se inició el desarrollo de la nueva plataforma, en ella se siguieron manteniendo patrones de diseño correspondientes a la versión previa, motivado en parte porque el soporte de patrones de diseño enfocados a la orientación a objetos aún no estaba plenamente desarrollado en el lenguaje de programación PHP.

Este diseño previo se analizará en el siguiente apartado, pero, a modo de análisis rápido, se puede decir que el diseño existente complicaba enormemente la capacidad de extender las funcionalidades de la plataforma y el trabajo colaborativo entre varios programadores. Dado que Tutor era, y es, un sistema en crecimiento, y que de un tiempo a esta parte han sido varios los programadores que han trabajado y trabajan en la plataforma, se hacía necesario dotar al sistema de cierta modularidad, para que cada uno de los programadores pudiera desarrollar su trabajo de forma independiente, sin que la integración del trabajo de todos ellos supusiera un quebradero de cabeza.

Dicho de otro modo, si todos los desarrolladores tienen que trabajar y realizar modificaciones sobre los mismos archivos, es de suponer que tarde o temprano van a surgir conflictos a la hora de actualizar estos archivos. El objetivo, por tanto, es minimizar al máximo la posibilidad de que surjan estos conflictos.

2.2. Análisis del diseño previo del sistema

Una vez que, en el punto anterior, ha quedado claro por qué es necesario aplicar un diseño modular para un mejor desarrollo de la plataforma, a continuación analizaremos en profundidad los diferentes puntos que deberían ser mejorados con respecto al diseño existente en Tutor. Son los siguientes:

Concentración en único fichero de varias funciones no relacionadas

En gran medida, todas las funciones de Tutor se concentraban en cuatro ficheros: *student_func.php*, *teacher_func.php*, *admin_func.php* y *db_functions.php*. Los nombres hacen referencia al contenido del fichero, concentrando cada uno las funciones para estudiantes, profesores,… Esto implicaba que en un mismo archivo se podían encontrar funciones totalmente dispares, tales como las relacionadas con el control de asistencia o la zona de descargas, por ejemplo. Así, y siguiendo el principio de continuar el trabajo realizado, las nuevas funcionalidades de Tutor se iban añadiendo sobre estos archivos, complicando enormemente el trabajo si, por ejemplo, dos desarrolladores tenían que modificar ese mismo archivo.

"Código spaghetti"

Tutor está programado en su totalidad en PHP. El mal uso de este lenguaje por parte de desarrolladores ha propiciado que en muchos casos se asocie a este lenguaje con el conocido efecto *código spaghetti* [5]. Este problema surge cuando se entremezclan diferentes lenguajes, quedando éstos entrelazados. En el caso de Tutor, el entrelazado de código PHP, Javascript y HTML se produce en diferentes partes del sistema. Si bien a veces esta práctica es casi inevitable, el tratar de evitarla ofrece como resultado una programación más clara y un código más inteligible.

Escasez de clases y/o ausencia de una API del sistema

Extender un sistema implica hacer uso de las partes ya desarrolladas del mismo para dotarle de nuevas funcionalidades. En el caso de Tutor, ejemplos de este tipo de reutilizaciones de código serían: poder acceder fácilmente a los datos de un alumno, al listado de grupos de una asignatura, a los alumnos que pertenecen a un grupo,… No tiene sentido que cada desarrollador tenga que "*reinventar la rueda*" para desarrollar sus funcionalidades. Sin embargo, en Tutor no existían clases lo suficientemente desarrolladas como para evitar esto.

Fichero de lenguaje único

Tutor es un sistema multilenguaje (español e inglés, de momento, aunque se puede extender a otros idiomas), guardándose las diferentes cadenas de texto de cada funcionalidad en un único fichero. Una vez más, el concentrar todos estos textos en un único fichero complica el desarrollo colaborativo.

Múltiples puntos de entrada

PHP es un lenguaje interpretado en el que cada petición a una página genera una respuesta. En Tutor existían varios puntos de entrada, entendiéndose éstos como scripts que generan una salida HTML para el usuario. Es decir, el fichero *index.php* de la raíz del sistema genera una salida HTML. Sin embargo, el fichero *teacher_func.php*, por ejemplo, no genera ninguna salida, pues sólo contiene funciones que deben ser llamadas. Estos puntos de entrada deben encargarse de gestionar la autenticación del usuario, la selección de lenguaje y otras

tareas comunes a cualquier funcionalidad. Por tanto, al haber varios puntos de entrada, se tiene que estar repitiendo código en todos ellos.

En la versión previa de Tutor había seis puntos de entrada al sistema, ya que, además de la propia página principal, mostrada en la Figura 4, cada uno de los scripts que generaban los cinco botones del menú superior (Funciones, Asignaturas, Profesores, Sobre Tutor y Contáctenos) eran también puntos de entrada al sistema.

Figura 4 - Puntos de entrada de Tutor (indicados en rojo)

Estructura de directorios

Al igual que con el problema de los ficheros de funciones, la estructura de directorios del sistema no separaba los ficheros de las diversas funcionalidades del sistema. Así, por ejemplo, la carpeta de plantillas HTML contenía las plantillas de todas las funcionalidades del sistema. Si bien esto no impide el buen funcionamiento del sistema, sí que hace poco intuitivo determinar a qué plantilla pertenece cada una de las funciones. Algo similar sucedía con los ficheros de funciones Javascript o con los ficheros de clases.

En la Figura 5 se aprecia esta estructura de directorios, donde los ficheros que agrupan todas las funciones se marcan en rojo y los puntos de entrada se marcan en verde.

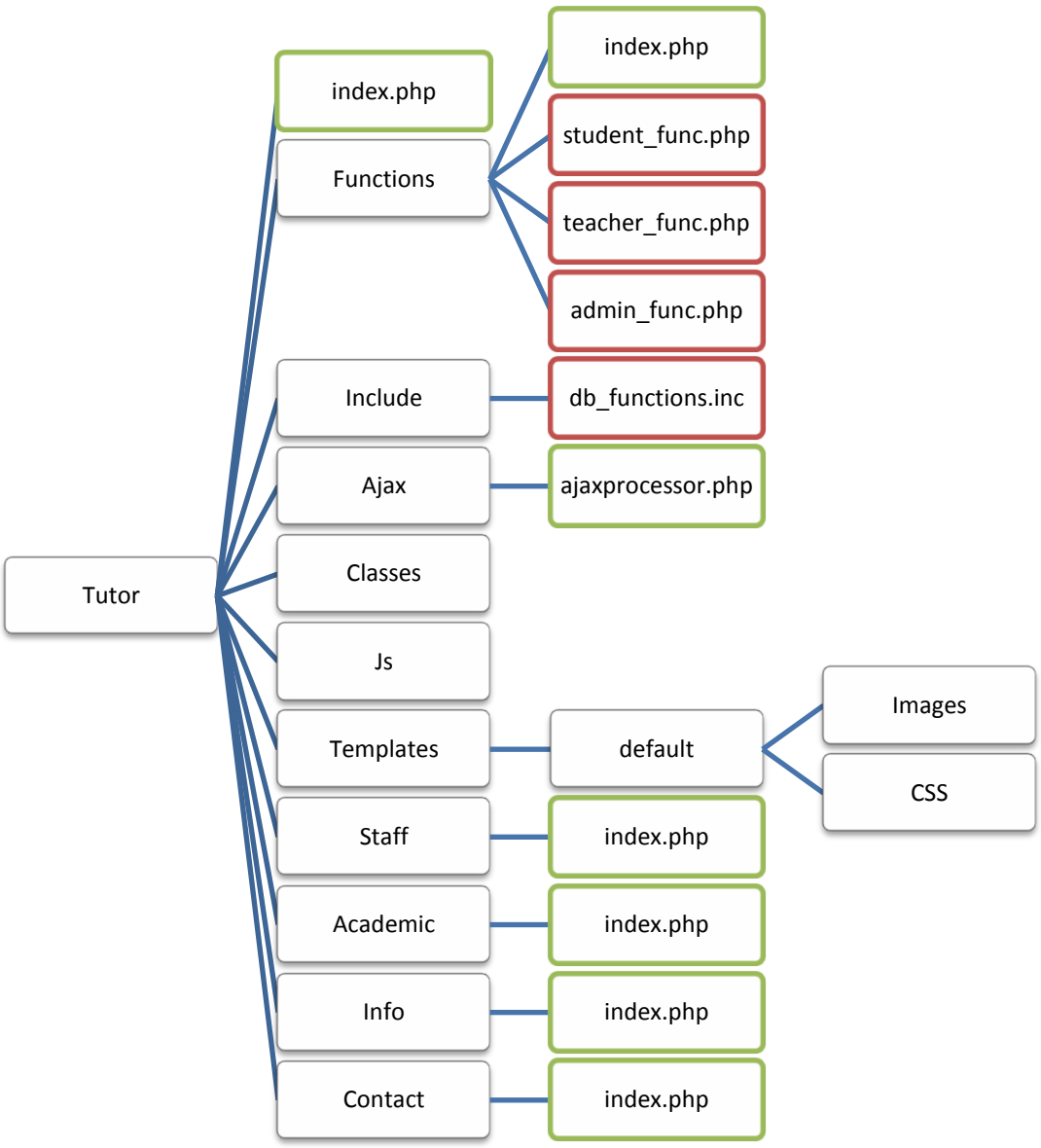


Figura 5 - Estructura de directorios de Tutor previa al diseño modular

En general, todos éstos son los principales puntos que dificultaban el trabajo simultáneo y el desarrollo modular de Tutor.

2.3. Especificación de requisitos del nuevo sistema

2.3.1 Requisitos no funcionales

Dadas las características e inconvenientes anteriormente mencionados, son muchas las cuestiones e ideas que surgen para paliar los problemas planteados y para mejorar el sistema en su conjunto. En cualquier caso, los principales requisitos no funcionales que debería cumplir el sistema con el nuevo diseño son las siguientes:

Soporte a un sistema modular

Que un desarrollador pueda añadir nuevas funcionalidades y éstas se adapten y sean completamente funcionales, sin tener que modificar partes del sistema ya existentes. Esto debería ser una de las claves del nuevo sistema, algo similar a una "autodetección" de módulos, de forma que la adición de un nuevo módulo sea automáticamente reconocida, sin tener que realizar modificaciones o enlaces en el sistema.

Separación modular del código

Identificar las diferentes áreas del sistema y separar éstas en diferentes ficheros y carpetas, que faciliten el mantenimiento del mismo.

Creación de una API básica

Dotar al sistema de una serie de funcionalidades y clases que faciliten tareas comunes es vital para que el sistema pueda ser extendido de forma sencilla y sin que futuros cambios en la estructura del mismo impliquen fallos en el funcionamiento del código generado.

Definición de una interfaz gráfica común

Para mantener la homogeneidad en cuanto a la interfaz gráfica del sistema se deben definir una serie de estilos y elementos (tablas, menús, botones, paneles desplegables,...) que sean reutilizables.

Retrocompatibilidad

Dado que en Tutor existen diferentes funcionalidades ya desarrolladas, y que algunas de ellas han sido implementadas por programadores que ya no están en el equipo de desarrollo, el nuevo diseño modular debería mantener la compatibilidad con las funcionalidades ya existentes.

Rendimiento

La detección y carga de nuevos módulos no debería producir grandes retardos en la generación de las páginas del sistema.

Seguridad

Aunque cada módulo puede tener diferentes características de seguridad, se debería intentar garantizar que un fallo de seguridad en un módulo no afecte al resto del sistema. Por ejemplo, un usuario identificado con rol de estudiante no debería poder acceder a datos de un usuario con rol de profesor bajo ningún concepto.

Extensibilidad del sistema

Añadir un número indeterminado de módulos no debería suponer un problema para la carga del sistema.

2.3.2 Requisitos funcionales

Dado el tipo trabajo que se explica en este capítulo, cuyo objetivo es rediseñar la estructura interna de un sistema, no habría por qué tener en cuenta nuevos requisitos funcionales. Sin embargo, se han tenido en cuenta dos de ellos:

Edición online de los menús del sistema

Dado que se pretende que un desarrollador no tenga necesidad de editar más que ficheros de su propio módulo, se hace necesario que, de alguna forma, el administrador del sistema pueda editar y reconfigurar, desde el propio sistema, los menús del mismo.

Edición online de los ficheros de lenguaje

Sería recomendable también que las diferentes cadenas del sistema pudieran ser editadas desde el propio sistema. Esto haría posible que un usuario no familiarizado con la programación pudiese corregir y traducir los diferentes textos del sistema.

El primer interrogante que surge tras enumerar estas características es: ¿Existe algún sistema en la actualidad que cumpla las mismas? La respuesta es clara: existen, y muchos. Redes sociales de código abierto, como Elgg [7], plataformas educativas como Moodle [35] y un largo etcétera de aplicaciones de otros tipos permiten a los desarrolladores extender funcionalidades mediante el desarrollo de módulos. Por otra parte, y a modo de ejemplo, Facebook [9] permite a los propios usuarios de la red traducir el sistema y votar las traducciones de otros usuarios, con el fin de elegir las más adecuadas. Por tanto, en mayor o menor medida, ambas características están implantadas en sistemas ya existentes.

A la hora de tratar la modularidad y extensibilidad, estos sistemas suelen usar una estructura de carpetas donde cada área o módulo está contenido en su propia carpeta, almacenando esta carpeta los diferentes ficheros requeridos por el módulo. Un ejemplo de esta estructuración se puede ver en el FAQ de Elgg [7].

En cuanto a interfaces gráficas se refiere, también está creciendo el fenómeno de usar librerías de terceros, como Yahoo UI [58] o jQueryUI [27].

2.4. Análisis y Diseño

Para dotar a Tutor de los requisitos anteriormente citados es necesario estudiar cuál es la opción más adecuada para resolver los problemas que se han planteado. Las preguntas y problemas a los que hay que dar respuesta son los siguientes:

- ¿Cuál es la estructura típica de un módulo en Tutor?
- ¿Qué estructura de directorios es la idónea para este diseño modular?
- ¿Qué clases y funciones son necesarias para gestionar esta nueva arquitectura?
- Mantener la retrocompatibilidad del sistema.
- Elegir, si existe, o crearla, si no existe, una interfaz gráfica que pueda ajustarse a las necesidades de Tutor.
- Medidas de seguridad aplicables al nuevo diseño.

Una vez que se tengan respuestas y soluciones para estas preguntas y problemas, y, por tanto, se haya definido el nuevo diseño, el siguiente paso será realizar la separación de las funcionalidades existentes en módulos.

Como es lógico, cualquier cambio aplicado a la base del sistema podría causar problemas no previstos en el futuro. Cabe recordar en este punto que Tutor es un **sistema en producción** y que ha sido desarrollado por varias personas. Por lo tanto, cabe extremar las precauciones para causar el mínimo inconveniente posible a los muchos usuarios que utilizan el sistema.

2.4.1 Estructura de un módulo

En el nuevo diseño de Tutor, se entiende que un módulo es cada uno de los diferentes fragmentos de código y archivos del sistema que componen una determinada funcionalidad o una serie de funcionalidades que están relacionadas. Por ejemplo, el módulo de *Control de Asistencia* estaría compuesto tanto por la gestión de sesiones por parte del profesor como por las funcionalidades de anotar asistencia y ver el historial de asistencias del alumno. Tanto éste como otros módulos se componen de una serie de archivos que podemos catalogar dentro de

las siguientes 7 categorías, aunque no necesariamente cada módulo ha de tener archivos de todas las categorías:

Archivos de funciones en PHP

Funciones que no forman parte de ninguna clase y que generan contenido final en HTML o son usadas por otras funciones para generar dicho contenido final.

Archivos de funciones AJAX en PHP

Funciones que responden a peticiones de tipo AJAX [1] o asíncronas, generando contenido HTML o JSON [30], o efectuando algún cambio en el sistema sin que sea necesario una recarga de la página.

Archivos de definición de clases en PHP

Estos archivos contienen definiciones de las diferentes clases que componen el sistema. Un archivo sólo contendrá la definición de una clase, por motivos que se analizan más detalladamente en el apartado correspondiente al *Cargador de Clases*, 2.5.8.

Archivos de funciones en Javascript

Funciones que contienen el código Javascript del sistema, el cual se encarga en la mayoría de los casos de aumentar la usabilidad y de realizar operaciones del sistema en el lado del cliente.

Plantillas en HTML

Archivos que contienen las plantillas HTML usadas en las diferentes funciones del sistema.

Hojas de estilo CSS

Archivos que contienen las hojas de estilo CSS usados en los diferentes módulos del sistema. Cada módulo puede tener una serie de elementos únicos.

Imágenes

Imágenes usadas en el módulo. Al igual que con las hojas de estilo, un módulo puede tener imágenes que sólo sean usadas en el mismo.

Delimitadas todas las categorías, se puede crear una estructura de carpetas que contenga todos los ficheros que se comparten en un módulo, quedando gráficamente como se muestra en la Figura 6, teniendo cada directorio el siguiente significado:

/functions/: Carpeta para los ficheros de funciones en PHP.

/ajax/: Carpeta para los ficheros de funciones AJAX en PHP.

/classes/: Carpeta para los ficheros de definiciones de clases en PHP.

/js/: Carpeta para los ficheros de funciones en Javascript.

/templates/: Carpeta para las plantillas HTML.

/templates/images/: Carpeta para las imágenes.

/template/css/: Carpeta para las hojas de estilo CSS.

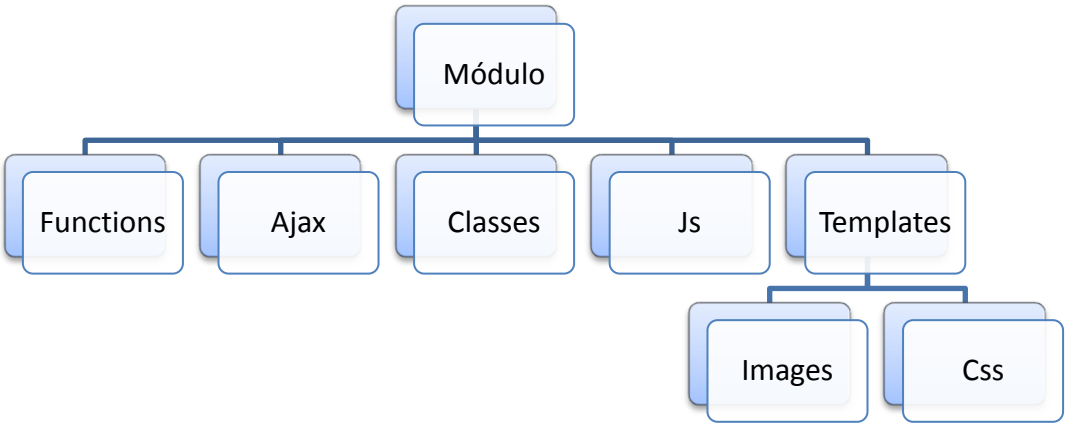


Figura 6 - Estructura de directorios de un módulo

2.4.2 Diagrama de clases

Tras el análisis realizado, el diagrama de clases propuesto para afrontar este nuevo diseño es el que se muestra en la Figura 7. En este diagrama, la clase *Tutor* representa el núcleo del sistema, relacionándose directamente con las clases *FunctionManager* y *LanguageManager*, responsables de gestionar las funcionalidades y los idiomas disponibles en el sistema respectivamente. También se asocia con cada uno de los módulos del sistema, los cuales pueden ser del tipo *GenericModule*, si no implementan nada nuevo respecto a la clase *AbstractModule*, o de cualquier otro tipo, si implementan su propia versión de ésta. Las clases *GradesModule, NoticesModule*, *PMModule*, *PreferencesModule*, *WorksModule*, *DownloadsModule* y *AssistancesModule* son una especialización de la clase *AbstractModule*, dado que, como se explica en el Capítulo 5, necesitan implementar su propia función para el cálculo de las correspondientes estadísticas.

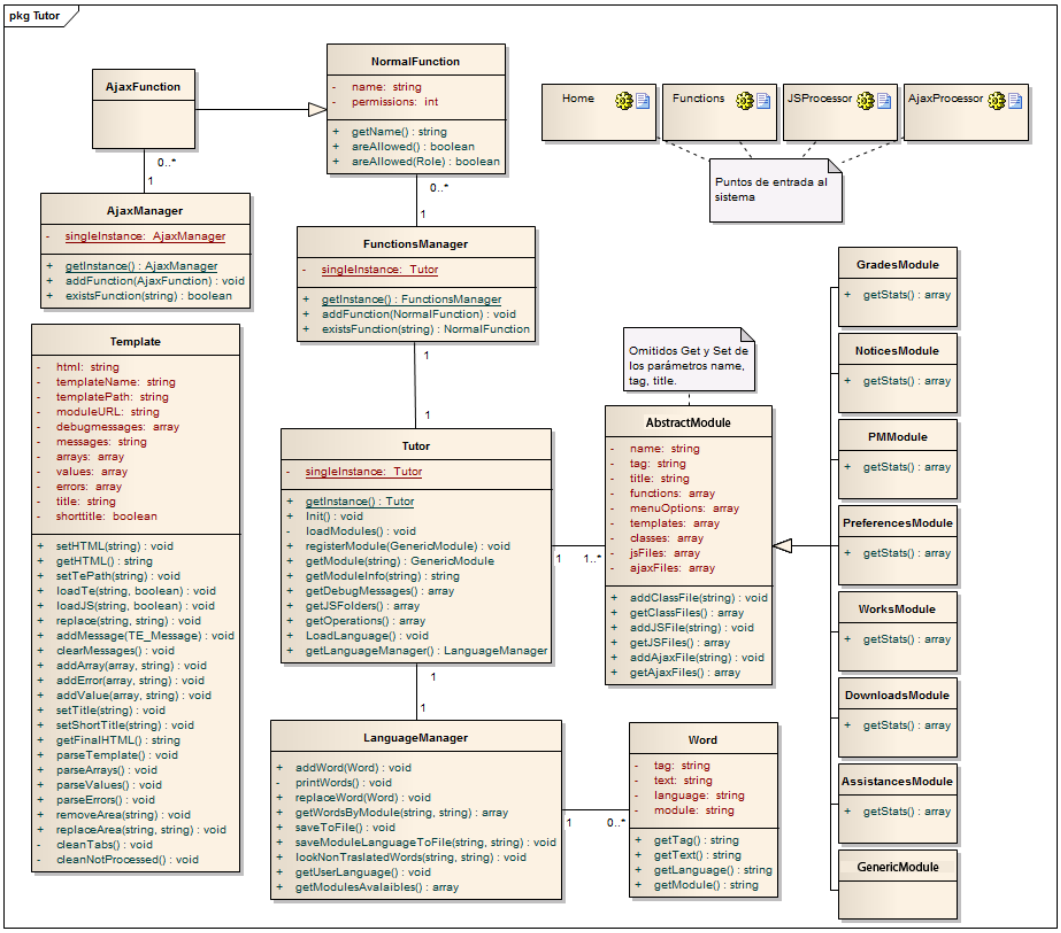


Figura 7 - Diagrama de clases del nuevo diseño del sistema

La clase *AjaxManager* gestiona las funciones de tipo AJAX que registra cada uno de los módulos. No se relaciona directamente con la clase *Tutor*, pues los módulos registran sus propias funciones AJAX directamente en esta clase, sin pasar por la clase *Tutor*. Esto es así porque antes del nuevo diseño modular ya existían funcionalidades que hacían uso de funciones AJAX mediante una versión previa de este gestor. Este comportamiento se detalla en el apartado 2.5.4.

La clase *Template* se muestra desvinculada de cualquier otra clase porque no tiene ninguna relación directa. Sin embargo, se puede generar una nueva instancia de ella en cualquier parte del sistema. Se usa para generar contenido HTML, haciendo uso de las plantillas HTML que cada módulo incluye. De esta forma, se desvinculan las capas de controlador y vista del sistema, y se proporciona al desarrollador una forma sencilla de generar contenido HTML. Esta explicación se detalla en el apartado 2.5.3.

Por último, se representan los 4 nuevos puntos de entrada al sistema, que son:

- **Home**: Punto de entrada para la página principal (o de bienvenida) de Tutor (Figura 8). Se encarga de generar el contenido HTML correspondiente a esta página.

Figura 8 - Página de bienvenida de Tutor

- **Functions**: Punto de entrada para todas las funcionalidades de Tutor. Se encarga de responder a la solicitud del usuario de acceder a un menú o una funcionalidad del sistema. Para ello, analiza los parámetros *idmenu* e *idop* que acompañan a la solicitud, verifica los permisos del usuario respecto al menú o a la funcionalidad solicitada y genera la salida HTML correspondiente. Si el usuario tiene permiso para acceder a la funcionalidad requerida, se delega en el módulo que implementa dicha funcionalidad la generación del contenido HTML correspondiente. En caso contrario, se le muestra un mensaje de error. En la Figura 9 se puede ver el parámetro *idop* cuyo valor es *staff*, que corresponde al *Índice de profesores* de Tutor.

Figura 9 - Punto de entrada Functions

- **JSProcessor**: Punto de entrada a los ficheros Javascript de Tutor. A diferencia de otros sistemas, los ficheros Javascript usados en Tutor no se envían directamente al navegador, sino que se preprocesan para traducir cadenas de texto (si las tuvieran) con respecto al idioma elegido por el usuario o para añadir constantes del sistema, tales como URL completa al servidor o subcarpeta en el dominio que da acceso al sistema. Además, se minimiza la salida. El script responsable de este punto de entrada se describe con más detalle en el apartado 2.5.5.

- **AjaxProcessor**: Las peticiones AJAX al sistema se realizan mediante este punto de entrada, con el fin de evitar un punto de entrada diferente para las peticiones AJAX por cada uno de los módulos. El funcionamiento de este punto de entrada se detalla en el apartado 2.5.4.

2.4.1 Diseño por capas

La Figura 10 muestra el nuevo diseño por capas de Tutor. En la parte inferior izquierda de la figura aparecen el componente *Núcleo del sistema* sobre el componente *Base de datos*. Tan sólo el núcleo del sistema tiene acceso a la base de datos del mismo, evitando así que cada módulo defina su propia conexión a esta o que pueda efectuar consultas sin la supervisión previa del núcleo, lo que pondría en compromiso la seguridad de Tutor. Dentro del núcleo se encuentran las clases y funciones relacionadas con los usuarios y las asignaturas y grupos. Esto es así porque, a diferencia de los módulos, no tendría sentido hacer uso de Tutor si careciera de estos elementos. Es decir, se podría suprimir cualquier módulo del sistema, como por ejemplo, *Control de Asistencia* o *Zona de descargas*, pero el sistema seguiría siendo funcional. Sin embargo, si se suprimieran los usuarios o las asignaturas y grupos, el sistema quedaría inoperativo.

A la derecha de los componentes *Núcleo del sistema* y *Base de datos* se encuentran los componentes *Gestor de funciones*, *Gestor de lenguajes* y *Clase Template*. Estos componentes están disponibles de forma directa para los módulos, pero no acceden a la base de datos en ningún momento.

Por encima de los elementos anteriormente mencionados se encuentran cada uno de los *Módulos* del sistema. Cada módulo se divide en una arquitectura Modelo-Vista-Controlador. En la capa Modelo residen las *Clases*, las *Funciones* y los *Ficheros de lenguaje*. En la capa Vista-Controlador residen las *Plantillas HTML*, *Ficheros Javascript* y *Hojas de estilo CSS*. El motivo de que las capas Vista-Controlador estén ligadas surge de que, en Tutor, las plantillas HTML y los ficheros Javascript comparten aspectos de ambas capas. Las plantillas HTML pueden incluir botones en la capa de Vista que, mediante las funciones definidas en los ficheros Javascript modifiquen ésta, sin tener que hacer uso del Modelo. Por ejemplo, las tablas HTML mostradas en las funcionalidades de los diferentes módulos se pueden reordenar sin hacer uso del Modelo cuando un usuario solicita los datos ordenados de otro modo.

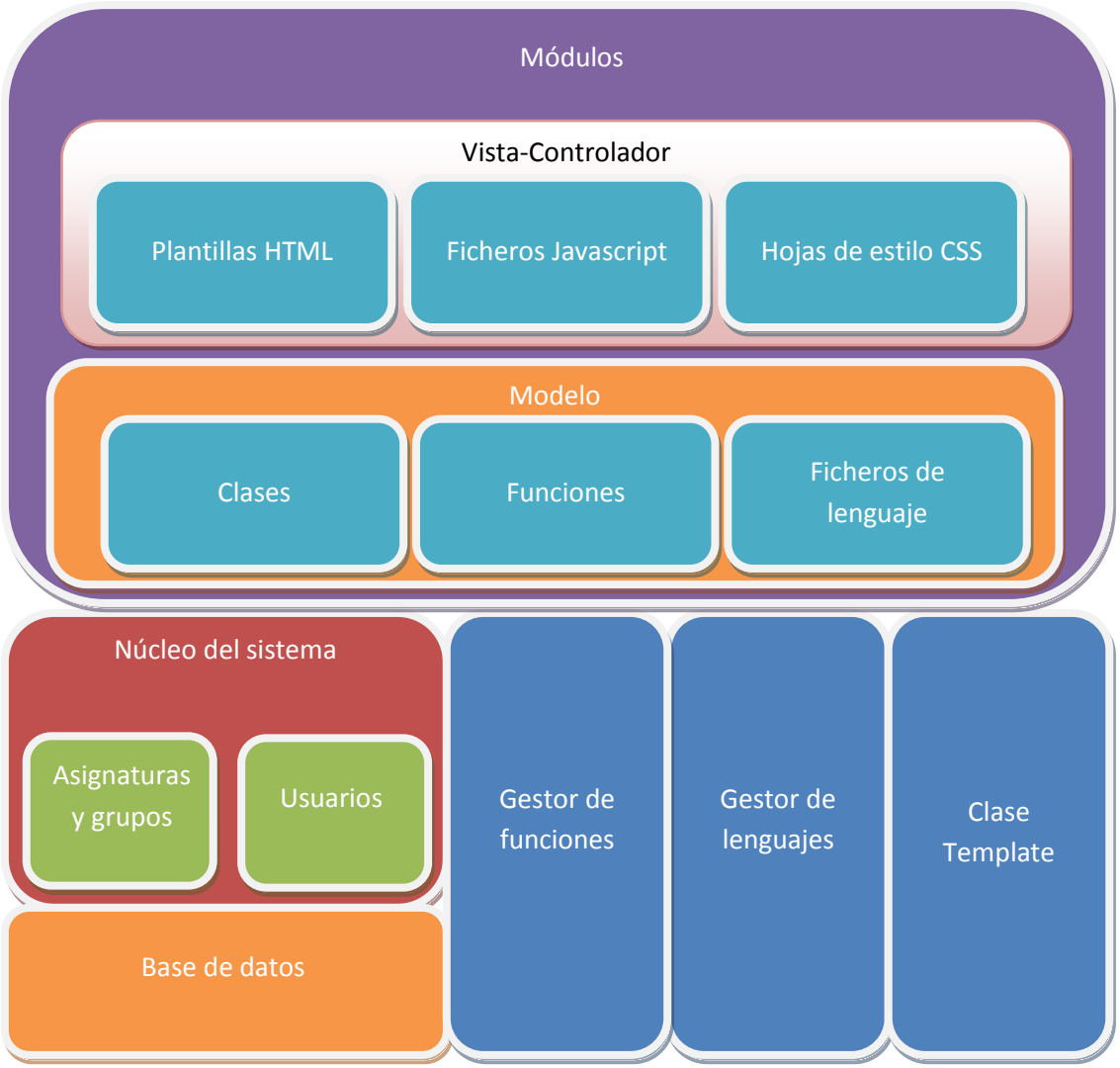


Figura 10 - Diseño por capas de la nueva arquitectura

2.5. Implementación

Para realizar la implementación correspondiente al nuevo diseño, se han añadido o modificado una serie de clases, módulos y otros elementos al sistema. A continuación se listan las principales modificaciones efectuadas, explicándose en qué consiste cada uno de estos cambios en los diferentes subapartados que forman parte de esta sección.

- Clase Tutor
- Clases AbstractModule y GenericModule
- Clase Template
- Cargador de clases
- Punto de entrada AjaxProcessor
- Punto de entrada JSProcessor

- Gestor de idiomas
- Gestor de menús
- Interfaz gráfica jQueryUI

2.5.1 Clase Tutor

La clase *Tutor* se ha implementado con el objetivo de inicializar el sistema y detectar y registrar los módulos existentes. Dadas las características de un sistema en PHP, en el que la aplicación se ejecuta para cada petición realizada, y dado que estas peticiones se hacen a través de los diferentes puntos de entrada esta clase será llamada en cada punto de entrada a Tutor. Se usará un patrón *Singleton* [39] para que se pueda instanciar de forma única a la clase desde las diferentes funciones que requieran un acceso a ella.

El proceso detección y registros de estos módulos se muestra en la Figura 11, en pseudocódigo.

```
FOR cada carpeta en la carpeta modules
        IF módulo tiene funciones THEN Incluir funciones
        IF módulo NO INSTANCIA SU PROPIA CLASE Module
                CREAR nueva instancia Module genérica
        IF módulo tiene clases/plantillas HTML/funciones
           AJAX/ficheros Javascript
                THEN Registrar elementos en la instancia Module
        IF tiene el módulo textos
                THEN Registrar textos en la variable global de textos
        y añadir textos en el gestor de lenguajes
        AÑADIR instancia Module en el sistema
FIN FOR
```

Figura 11 - Pseudocódigo para la carga de un módulo

2.5.2 Clases AbstractModule y GenericModule

La clase abstracta *AbstractModule* es la clase base que contiene las funciones para manejar la información común a todos los módulos. Es también la base para que cada módulo extienda esta clase con sus propias funciones de tipo común, pero, si un módulo no necesita extender la clase, podrá hacer uso de la clase *GenericModule*, la cual es una generalización vacía de la clase *AbstractModule*. El objetivo de este diseño es evitar tener que crear una clase específica que herede de *AbstractModule* para los módulos existentes, pues esto violaría el requisito de la retrocompatibilidad.

La clase *AbstractModule* contiene las siguientes propiedades para cada módulo:

- String name: Nombre identificativo (p.ej. GroupsManager).
- String tag: Etiqueta o abreviatura identificativa (p.ej. GM).
- String title: Título traducido al idioma actual (p.ej.: Gestor de grupos).

- array functions: Funciones públicas que implementa el módulo.
- array menuOptions: Opciones de menú disponibles.
- array templates: Índice con las plantillas incluidas.
- array classes: Índice con las clases incluidas.
- array jsfiles: Índice con los ficheros Javascript incluidos.
- array ajaxfiles: Funciones AJAX registradas.

Como se explicó en la sección 2.5.1, la clase *Tutor* es la responsable de completar los atributos *functions*, *templates*, *classes*, *jsfiles* y *ajaxfiles*. Para los atributos *name*, *tag* y *title*, el comportamiento es condicional, y pueden darse dos casos:

- El módulo implementa una clase que hereda de *AbstractModule*. En este caso, el módulo es el responsable de instanciar la clase hija, especificar los atributos *name*, *tag*, y *title*, y registrarla en el sistema.
- El módulo no implementa una clase que hereda de *AbstractModule*. En este caso, la clase *Tutor* crea una instancia de la clase *GenericModule* y efectúa su registro. Los atributos *name*, *tag* y *title* serán el nombre de la carpeta que alberga el módulo.

Mediante este comportamiento se permite la migración progresiva de los módulos existentes de Tutor a la nueva estructura, sin afectar a su funcionalidad, y se abre el camino para que los nuevos módulos implementen su propia clase.

Por otra parte, la clase *GenericModule* implementa los métodos asociados para cambiar los parámetros *name*, *title* o *tag,* y para añadir elementos o acceder a cada uno de los *arrays*.

2.5.3 Clase Template

La clase *Template* es la responsable de transformar las plantillas HTML de Tutor. Estas plantillas se componen de código HTML normal y un código propio de Tutor para el trato de variables, matrices, áreas y otra serie de funcionalidades. La carga de una plantilla se hace mediante la función *loadTe* de la clase:

```
$sheet = new Template();
$sheet->loadTe('downloads.te');
```

Dado que hasta ahora todas las plantillas se albergaban en la misma carpeta, */templates/default/*, la búsqueda de la plantilla era trivial. Sin embargo, con el nuevo diseño modular las plantillas estarán contenidas en diferentes carpetas. Para ***mantener la retrocompatibilidad*** el sistema seguirá buscando en la carpeta */templates/default/*, pero en caso de no encontrar la plantilla indicada, hará uso de la clase Tutor para buscar entre las plantillas registradas en el sistema. Esto implica que, de cara al futuro, las plantillas deberán tener un nombre único, lo cual se soluciona usando prefijos en el nombre de éstas.

Otro problema es el relacionado con que cada plantilla puede hacer referencia a archivos CSS o imágenes del propio módulo.

De cara a generar contenido HTML el desarrollador del módulo en sí no tiene porque conocer la ubicación final de una plantilla en el servidor, ni mucho menos la dirección donde se aloja el servidor. Para salvar este problema se define una nueva variable para las plantillas, $ModuleURL$, de cara a que el desarrollador de un módulo no tenga que preocuparse por la ruta donde se albergará el mismo.

Por ejemplo, en la plantilla del módulo de mensajería privada, la cual usa sus propios estilos CSS, haríamos referencia al fichero CSS de la siguiente forma:

```
<link href="$ModuleURL$pm.css" type="text/css"/>
```

Lo cual, una vez traducido por el procesador de plantillas, quedaría como se muestra a continuación:

```
<link href="/modules/private_messages/pm.css" type="text/css"/>
```

Este mismo proceso de aplicaría para las imágenes y demás elementos que deban ser referenciados, sin que el desarrollador tenga que preocuparse de en qué carpeta reside su módulo.

2.5.4 Punto de entrada AjaxProcessor

El punto de entrada AjaxProccesor es el encargado de responder a las peticiones AJAX [1] que se realizan en Tutor.

En general las peticiones en AJAX en Tutor se hacen mediante el uso de JQuery [28], aunque, en las funcionalidades donde el uso de AJAX no es suficiente, también se realizan peticiones mediante el uso de IFrames ocultos, según define el patrón AJAX IFrame Call [19].

En el primero de los casos la petición AJAX es recibida y procesada por el script *ajaxprocessor.php*. Para ello toda petición AJAX incluye una variable de nombre *get*, cuyo valor hace referencia a la función que debe responder a la petición. En general la petición AJAX mediante JQuery se puede ver en la Figura 12:

```
$.ajax({url: MainURL+"ajax/ajaxprocessor.php",
                type: "post",
                dataType: "json",
                data: {
                        get: "getFilesList",
                        subject: $("select#subjectid").val(),
                        anything: "1",
                        academicyear: $("select#academicyearid").val(),
                        groupid: $("select#groupid").val()
                },
                async: true,
                timeout: 10000,
                success: function(response){…}
                …
```

Figura 12 - Petición AJAX mediante JQuery

En concreto, esta petición solicita los ficheros para la *Zona de Descargas* de una asignatura. El parámetro *get*, resaltado en amarillo en la Figura 12, y cuyo valor es *getFilesList*, es el que indica al procesador de peticiones qué función debe encargarse de gestionar éstas. Antes del diseño modular, esta asignación se realizaba mediante una estructura *switch* definida en el propio *ajaxprocessor.php* y que asociaba a cada valor de *get* una función del sistema.

Con el nuevo diseño se pretende que cada módulo pueda registrar sus propias funciones AJAX. Para ello, se han desarrollado dos clases, *AjaxManager* y *AjaxFunction*:

AjaxManager

Esta clase está implementada mediante el patrón Singleton [39]. Se encarga de gestionar la lista de funciones AJAX del sistema e implementa sólo dos operaciones:

- void addFunction(AjaxFunction function): Registra la función function.
- AjaxFunction existsFunction(string nameFunction): Devuelve la función nameFunction, si ésta existe en el sistema, o null, en caso contrario.

AjaxFunction

Esta clase representa a una función AJAX del sistema. Sólo contiene dos variables, *name*, con el nombre de la función AJAX que representa, y *permissions,* una variable binaria que, al estilo de los permisos en los sistemas de ficheros UNIX, permite asignar permisos a los diferentes roles del sistema.

Mediante estas dos clases, la responsabilidad de registrar una función AJAX recae sobre el módulo. Dicho registro se realizará de la forma indicada en la Figura 13.

```
$ajaxManager = AjaxManager::getInstance();
$ajaxManager->addFunction( new AjaxFunction ("getStudentFilesList",
_STUDENTS_ ));
$ajaxManager->addFunction( new AjaxFunction("getFilesList", _TEACHERS_ |
 ADMINS  )
```

Figura 13 - Registro de función AJAX en el sistema

Cuando el script *ajaxprocessor.php* debe responder a alguna petición de función AJAX, instancia a *AjaxManager* y le pregunta por ésta. Si la función existe, se ejecutará. Si no, se devuelve un mensaje de error como salida a la petición AJAX, indicando que no se encontró la función buscada.

El segundo parámetro del constructor de la clase *AjaxFunction*, el parámetro *permissions*, está pensado para indicar qué roles de usuarios pueden hacer uso de una función AJAX. Así, por ejemplo, una función AJAX registrada indicando como permisos "TEACHERS |ADMINS" indica que sólo podrán hacer uso de esa función los usuarios con rol de profesor o administrador. De este modo, si se registra con estos permisos la función AJAX *EliminarFichero*, aunque esta función tuviese algún problema de seguridad interno, un usuario de tipo alumno no podría explotarlo, porque el propio sistema le negaría el acceso a la función.

2.5.5 Punto de entrada JSProcessor

Tutor hace uso del fichero *js.php* para obtener los diversos ficheros Javascript que puedan ser requeridos por las diversas plantillas de los diferentes módulos. Traducido a código, esto significa que una inclusión de un fichero Javascript tradicional en HTML:

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

Se realiza de la siguiente forma:

```
<script type="text/javascript" src="/js/js.php?js=ajax_functions"></script>
```

Esto se realiza así para permitir la utilización de etiquetas de lenguaje (Ver apartado 2.5.6. Gestor de idiomas).

De cara al diseño modular, se ha modificado este fichero para que, al igual que se hizo con la clase *Template*, busque inicialmente el fichero indicado en la carpeta /js/ y, en caso de no encontrar el fichero indicado, busque entre los ficheros Javascript añadidos por los módulos.

Adicionalmente a esta mejora, se ha aprovechado para incluir un minimizador de código Javascript, concretamente JS Min [18]. Este minimizador reduce el código Javascript, eliminando comentarios, espacios innecesarios, saltos de línea,…, con el fin de reducir la cantidad de datos a ser enviados desde el servidor.

2.5.6 Gestor de idiomas

La gestión de lenguajes de Tutor recae en gran parte sobre el procesador de plantillas. En las plantillas se pueden añadir elementos del tipo:

```
<h1>$t helloworld$</h1>
```

El procesador de plantillas reemplaza los elementos del tipo *$t_ETIQUETA$* por el valor de una matriz global llamada *$stringsLC*. Esta matriz se definía, hasta ahora, en los ficheros */language/es.php* y */language/en.php*, conteniendo los fragmentos de texto de cualquier función. De cara al nuevo diseño modular, cada módulo debería poder añadir sus etiquetas de forma independiente. Para mantener la retrocompatibilidad, la interoperabilidad entre módulos, en los casos necesarios, y de cara a poder utilizar textos que son generales a todo el sistema (palabras tales como "Nombre", "Apellidos",…) sin tener que redefinirlas en cada uno de los módulos se ha optado por mantener la matriz global *$stringsLC*, pero añadiendo a esta matriz las palabras definidas por cada módulo.

En cualquier caso, otro de los requisitos de la adaptación modular era poder permitir la edición de textos desde el propio sistema. Y para ello no es suficiente con añadir los textos en un lenguaje a la matriz global *$stringsLC*, sino que se requiere relacionar cada texto con su etiqueta o *tag*, el módulo que lo define y sus diferentes traducciones. Con este objetivo se han desarrollado las clases *LanguageManager* y la clase *Word*.

LanguageManager es un gestor que define, entre otras, las siguientes funciones:

- addWord(palabra): Añade una palabra a un módulo del sistema.
- replaceWord(palabra): Reemplaza una palabra existente.
- getWordsByModule(módulo): Devuelve las palabras de un módulo en concreto.
- saveModuleLanguageToFile(módulo, lenguaje): Guarda las palabras de un módulo y un lenguaje en el fichero correspondiente.
- lookNonTraslatedWords(lenguajeOrigen, lenguajeDestino): Busca las palabras no traducidas de un lenguaje respecto a otro.

Este gestor se ha usado para desarrollar un módulo de traducción, del cual se incluye una captura en la Figura 14. Este módulo permite que un usuario traduzca el texto que se muestra en las distintas interfaces de usuario del sistema, incluso sin tener conocimiento alguno del código interno del sistema, mediante una utilidad con una interfaz bastante sencilla e intuitiva. Aunque el módulo se ha diseñado inicialmente pensando en el inglés, de cara a un futuro se puede usar para añadir más lenguajes al sistema.

Para realizar esta adaptación sólo habría que editar la plantilla HTML que hace de interfaz de la utilidad, añadiéndole un campo adicional al código, pues todas las funciones de la clase *LanguageManager* están parametrizadas para recibir el lenguaje del que se desea añadir, traducir o reemplazar una palabra.

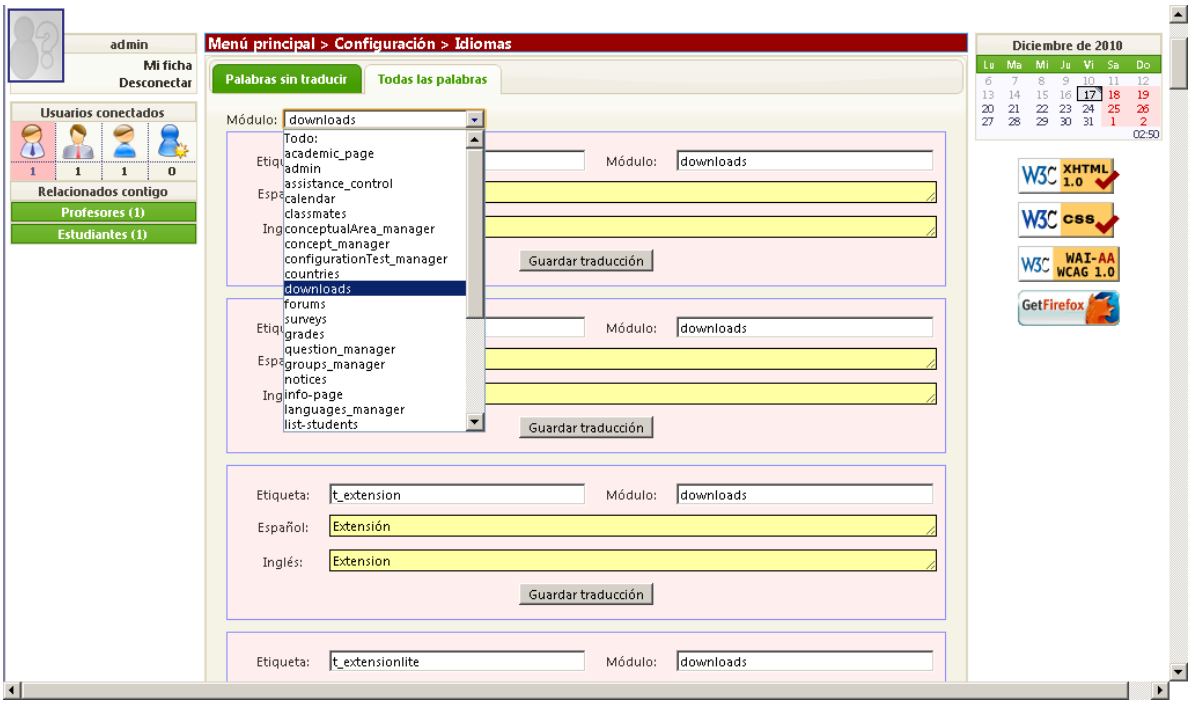


Figura 14 - Módulo de gestión de idiomas de Tutor

2.5.7 Gestor de menús

Mediante este módulo, el administrador puede editar y cambiar la estructura de opciones y subopciones del menú principal al que tienen acceso cada uno de los usuarios autenticados en el sistema, teniendo en cuenta que el menú de opciones que se presenta a cada usuario es distinto, dependiendo del rol que éste desempeñe (estudiante, profesor o administrador). Antes del cambio a diseño modular, las entradas de cada menú estaban almacenadas en un fichero de código, por lo que cada vez que un desarrollador quería añadir o eliminar alguna funcionalidad, debía editar este fichero.

Para solucionar esto, se ha rediseñado el sistema de menús, almacenándolo en la base de datos del sistema. Esto permite, por ejemplo, que un usuario de tipo estudiante o profesor pueda tener acceso a la opción del *Gestor de lenguajes*, con el objetivo de colaborar en la traducción de Tutor.

También permite situaciones como las de un desarrollador que ha integrado una funcionalidad en la versión de producción de Tutor, pero que aún no quiere que la funcionalidad esté disponible para todo el mundo, siendo ésta sólo visible para los usuarios que él indique.

La interfaz de usuario de esta funcionalidad puede verse en la Figura 15, en la que se muestra, en modo de edición, el menú para los usuarios de tipo profesor. El menú que el sistema le muestra al profesor a partir del menú generado en esta funcionalidad puede verse en la Figura 16.

Figura 15 - Gestor de menús

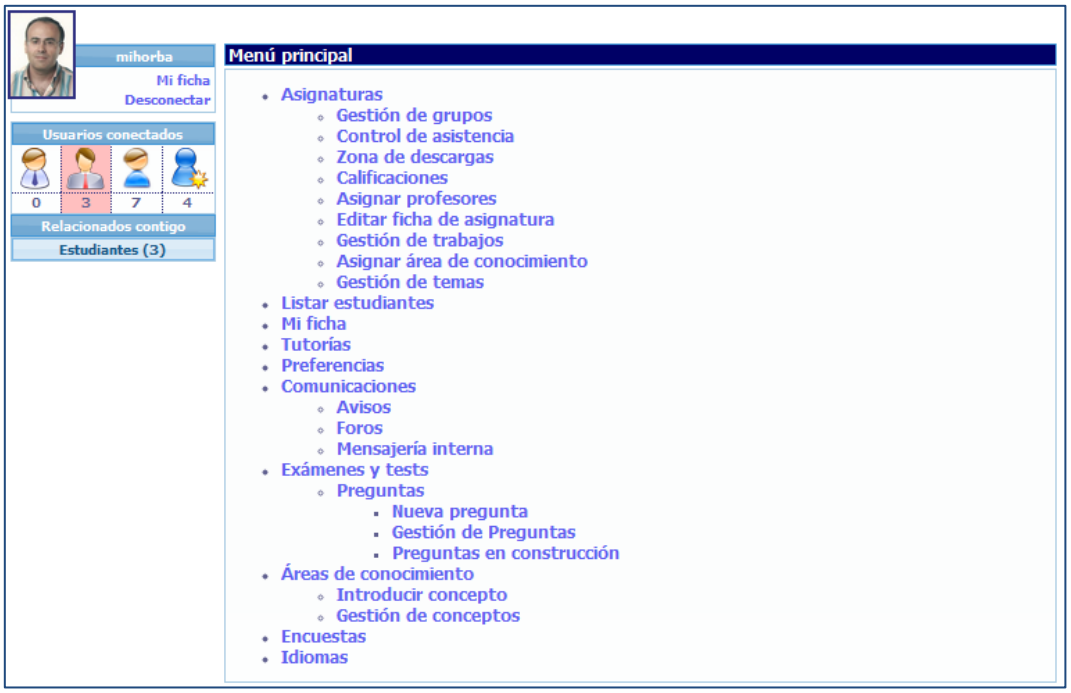


Figura 16 - Menú principal para el rol de usuario Profesor

2.5.8 Cargador de clases

Dado que cada módulo define sus propias clases y, de cara a cumplir los requisitos establecidos, éste no tiene porqué saber donde están localizadas esas clases, se hace necesaria una solución que, de forma transparente, permita al desarrollador instanciar sus clases sin preocuparse de que hayan sido incluidas o no.

En PHP, de forma similar al uso de sentencias *include* en C, es necesario haber incluido el archivo que define una clase para poder hacer uso de ella. Por tanto, parece casi ineludible que el desarrollador deba requerir al principio de sus scripts las clases que le vayan a ser necesarias.

Se podría también solucionar este problema incluyendo todas las clases conforme éstas se fueran registrando en el sistema, pero sería ineficiente tener la definición de todas las clases en memoria cuando en la mayoría de los casos se usarán sólo unas pocas de ellas.

Afortunadamente, desde PHP5 se proporciona un mecanismo de autocargado de clases que permite interceptar cuándo el desarrollador intenta invocar una clase que aún no ha sido definida. La función *__autoload* de PHP será llamada cuando se produzca esta situación. Combinando esta función con el registro de las clases definidas por cada módulo en la clase *Tutor* se consigue la solución al problema, mostrada en la Figura 17.

```
        function __autoload( $className ) {
                $classPath = _TUTORPATH_ . "class\class_" . $className . '.php';
        if ( file_exists( $classPath ) ) {
                        require_once ( $classPath );
                return true;
                } else {
                        $Tutor = Tutor::getInstance();
                if (!is_null($Tutor->classesRegistered[$className]) ) {
        require_once($Tutor->classesRegistered[$className]);
        return true;
}
        }
        echo "Error loading $className";
        return false;
}
```

Figura 17 – Autocargado de clases

De esta forma, se sigue el patrón usado para las plantillas HTML y los ficheros Javascript. Si la clase existe en la carpeta base de clases de Tutor, se cargará la definición desde allí. En caso contrario, se buscará si la clase ha sido registrada en el sistema previamente por algún módulo, en cuyo caso se cargará la definición dada por dicho módulo.

2.5.9 Interfaz gráfica jQueryUI

Dado que la librería que se usa en Tutor para las funcionalidades que se desarrollan mediante Javascript es JQuery, y que muchos de los plugins disponibles para esta librería hacen uso de la extensión de la misma conocida como jQueryUI, adaptar y estandarizar Tutor al uso de esta librería era la opción más recomendable a la hora de definir una interfaz gráfica común para todos los módulos y funcionalidades existentes en Tutor.

En la Figura 18 se puede ver una muestra de los elementos de interfaz que proporciona jQueryUI. Los principales elementos definidos por la interfaz son:

- Paneles en forma de “acordeón”, esto es, conjunto de paneles donde sólo uno de ellos puede estar desplegado
- Paneles divididos en pestañas
- Ventanas o cajas de diálogo (Pop-Ups)
- Botones
- Campos de autocompletado
- Deslizadores (Sliders)
- Selectores de fecha (Datepicker)
- Barras de progreso

<

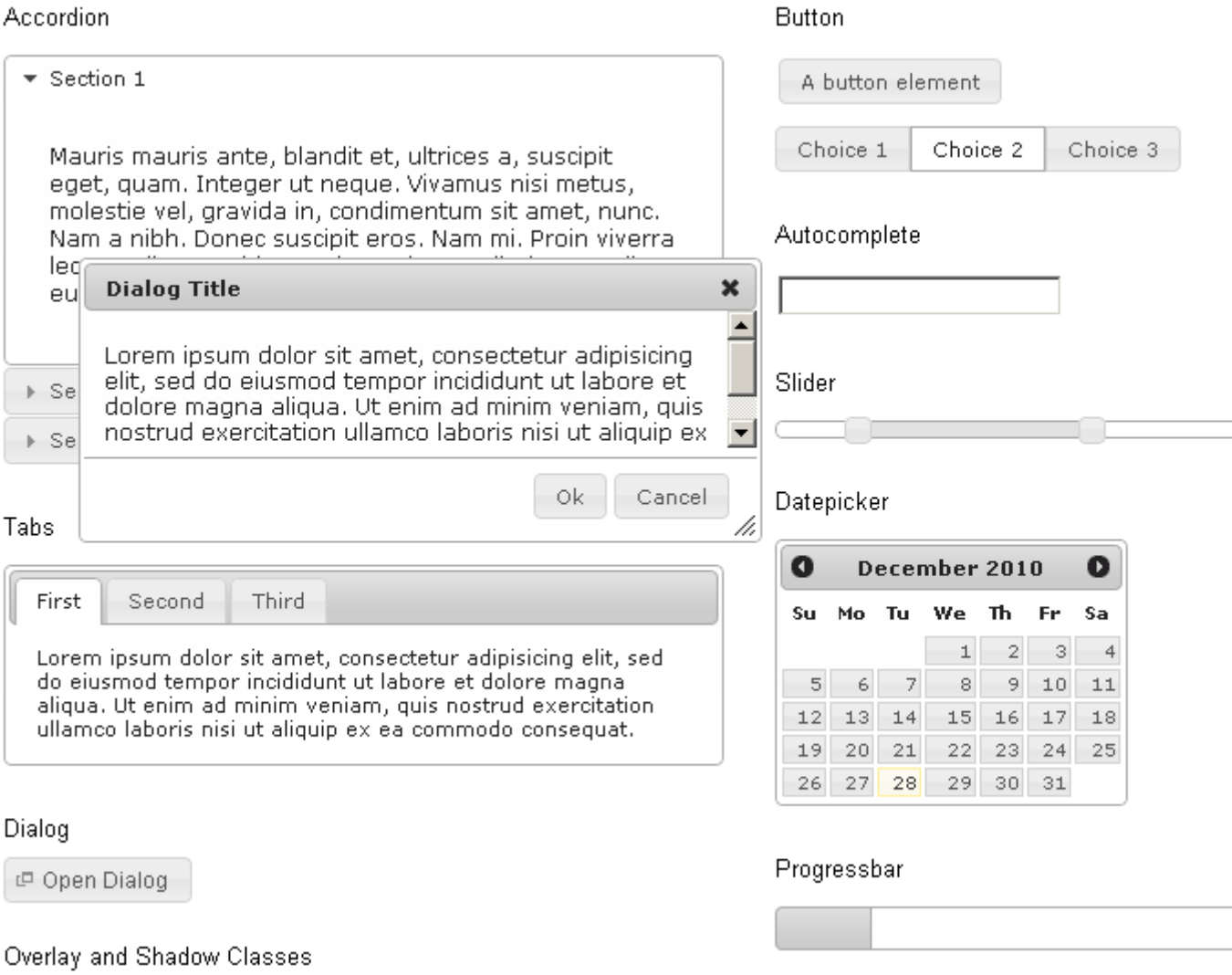


Figura 18 - Galería de elementos de interfaz de usuario en jQueryUI

Otra ventaja de usar jQueryUI es que integra una serie de temas, los cuales también son personalizables mediante jQueryUI *ThemeRoller* [29]. Estos temas consisten en combinaciones de colores y elementos gráficos que permiten modificar el aspecto de los elementos de la interfaz de usuario.

Para que el usuario de Tutor pueda aprovechar el grado de personalización que permiten estos temas, se ha añadido un nuevo selector de tema en la funcionalidad de preferencias del usuario. En la Figura 19 y la Figura 20 se puede ver el selector de tema con los 23 temas disponibles en la actualidad. Mediante la selección de cualquiera de ellos, el usuario puede previsualizar cómo varía el aspecto de la interfaz gráfica de Tutor según el tema escogido.

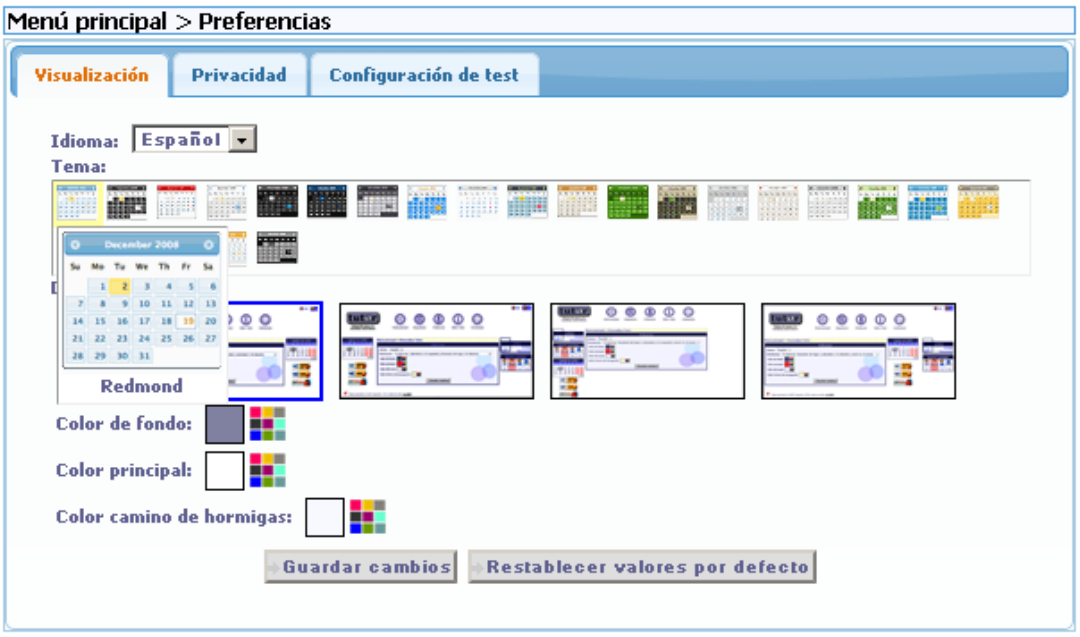


Figura 19 - Panel de preferencias de usuario usando el tema Redmond de jQueryUI

Figura 20 - Panel de preferencias de usuario usando el tema Sunny de jQueryUI

Para la adaptación de la interfaz de un módulo a jQueryUI, se debe hacer uso de una lista de definiciones de estilo CSS. El listado completo de estos estilos se encuentra en la documentación de la API de *jQquery UI CSS Framework* [26].

2.6. Módulos adaptados

Una vez realizados los cambios mencionados en la implementación, el siguiente paso era el de separar las clases, funciones, plantillas, ficheros Javascript, etc., en diferentes ficheros

para cada uno de los módulos. El número de módulos adaptados asciende a 27, y las funcionalidades que engloba cada uno de ellos son las siguientes:

- **AcademicPage**: Funcionalidad *Asignaturas* del menú superior de Tutor. Antes de aplicar el nuevo diseño, esta funcionalidad era un punto de entrada.
- **Admin**: Funcionalidades exclusivas del Administrador, tales como Añadir Profesor, Añadir Asignatura,...
- **AttendanceControl**: Funcionalidades relativas a *Control de Asistencia*, *Anotar asistencia* y *Ver historial de asistencias*.
- **Classmates:** Funcionalidad que representa el cuadro *"Relacionados contigo"* que se muestra debajo del cuadro de número de usuarios conectados en cada momento al sistema, y que se ofrece como mecanismo de conciencia de grupo.
- **ConceptualAreaManager**: Gestor de áreas conceptuales, que permite agrupar diversas asignaturas relacionadas por su temática y compartir información entre ellas.
- **ConceptManager**: Gestor de conceptos, que se usan en el módulo anterior.
- **ContactPage**: Funcionalidad *Contáctenos* del menú superior de Tutor. Previamente al nuevo diseño, era otro punto de entrada.
- **Downloads**: Funcionalidad de *Zona de descargas*.
- **Forums**: Funcionalidades relacionadas con los *Foros*.
- **Grades**: Funcionalidades relacionadas con las *Calificaciones*.
- **GroupsManager**: Funcionalidad del *Gestor de grupos***.**
- **InfoPage**: Funcionalidad que muestra la información *Sobre Tutor* del menú superior de Tutor. Previamente al nuevo diseño, era otro punto de entrada.
- **LanguagesManager**: Funcionalidad del *Gestor de idiomas*, descrito en el apartado 2.5.6.
- **ListStudents**: Funcionalidad de *Listar Estudiantes*.
- **MenuManager**: Funcionalidad del *Gestor de menús*, descrito en el apartado 2.5.7.
- **Notices**: Funcionalidades relacionadas con los *Avisos*.
- **Preferences**: Funcionalidad de *Preferencias* del usuario.
- **PrivateMessages**: Funcionalidad de la *Mensajería interna*, desarrollada en el Capítulo 3.
- **QuestionManager**: Funcionalidad del *Gestor de preguntas*, para la confección de exámenes.
- **StaffPage**: Funcionalidad *Profesores* del menú superior de Tutor. Previamente al nuevo diseño, era otro punto de entrada.
- **Statistics**: Funcionalidad de *Estadísticas, desarrollada en el Capítulo 5*.
- **SubjectProfiles**: Funcionalidad de *Ficha de asignatura*.
- **Surveys:** Funcionalidad de *Encuestas*.
- **TestManager**: Funcionalidad de *Tests*.
- **UserProfiles**: Funcionalidades de *Ficha de profesor* y *Ficha de estudiante*.

- **Works**: Funcionalidades de *Gestor de trabajos* y *Subir trabajos.*

Capítulo 3. Mensajería interna

3.1. Motivación

En un sistema al que se le pretende dotar de funcionalidades características de una de red social, no puede faltar un sistema que permita la comunicación entre usuarios. En el momento de comenzar el desarrollo de este proyecto existían dos funcionalidades en Tutor que facilitaban la comunicación entre usuarios, los *avisos* y los *foros*.

En el caso de los avisos se les da a los profesores la posibilidad de poner mensajes de texto en la plataforma, notificando a los alumnos de dichos mensajes mediante un indicador en la pantalla principal del sistema. Este método de comunicación permite la misma tan solo unidireccionalmente, de profesor a alumnos, o de profesor a profesores, pues en ningún momento se le brinda al alumno la posibilidad de responder a un aviso, tan solo se le da la oportunidad de leerlo. Por tanto la posibilidad de responder a un mensaje de este tipo queda relegada a medios tradicionales y externos a la plataforma, como puede ser el correo electrónico.

Figura 21 - Captura de la funcionalidad Avisos

Por su parte, la funcionalidad de los *foros* brinda tanto a profesores como alumnos la posibilidad de escribir un mensaje o responder a uno ya existente. Mediante esta funcionalidad si se permite una comunicación bidireccional, pues cualquiera puede iniciar una conversación o participar en una ya existente.

Sin embargo esta funcionalidad no permite escoger los destinatarios de los mensajes dado que automáticamente estos son dirigidos a todos los miembros de un grupo o de una asignatura. Del mismo modo tampoco permiten establecer comunicaciones entre dos únicos usuarios, como sería el caso de un alumno que consulta una duda a un profesor.

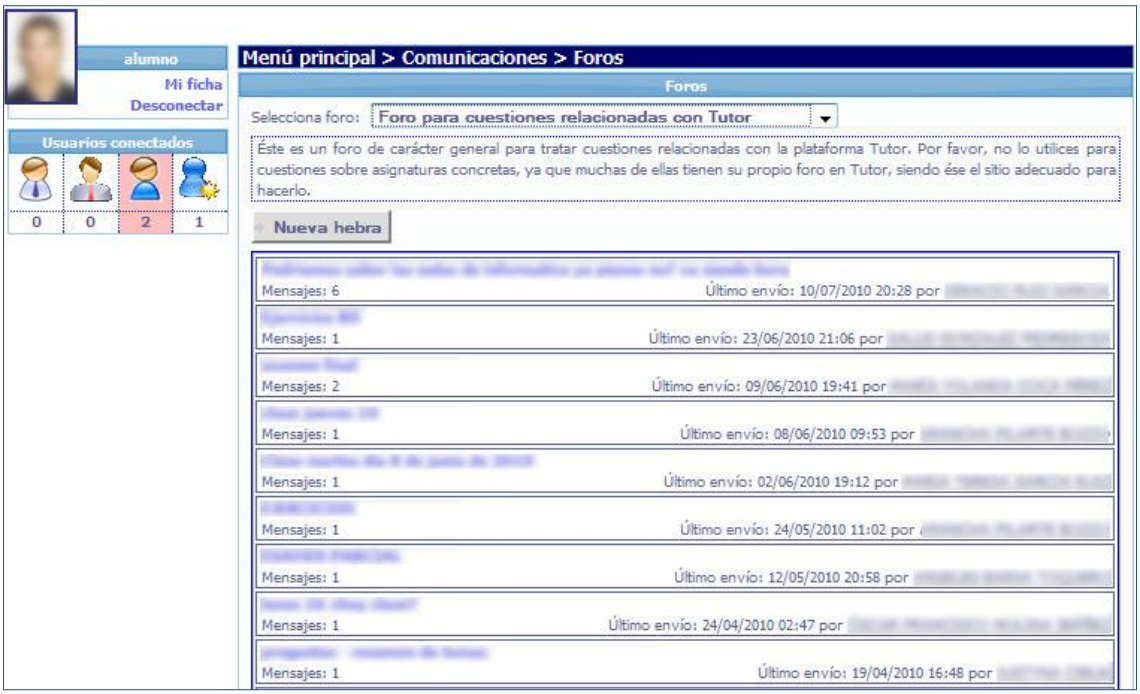


Figura 22 - Captura de la funcionalidad Foros

En conclusión podemos decir que las dos funcionalidades orientadas a la comunicación de Tutor previas a este proyecto no suplen dos requisitos muy importantes, la bidireccionalidad y la privacidad de la comunicación. Se hace por tanto obvio la necesidad de una nueva funcionalidad que complemente a las existentes y que palie estas carencias.

3.2. Estudio de herramientas de mensajería en redes sociales

A la hora de desarrollar este módulo era clave estudiar cómo se afronta la mensajería entre usuarios actualmente en las redes sociales más importantes. En mi opinión, en el mundo de la ingeniería es importante tratar de no reinventar la rueda, sino mejorar la existente. Con este objetivo realice un estudio de las dos redes sociales a nivel nacional e internacional más importantes, Facebook y Tuenti respectivamente, analizando 3 diferentes áreas: la bandeja de entrada, la visualización de los mensajes y la escritura de un nuevo mensaje.

3.2.1 Bandeja de entrada

La Figura 23 y la Figura 24 muestran las bandejas de entrada de los sistemas de mensajería de Tuenti y Facebook.

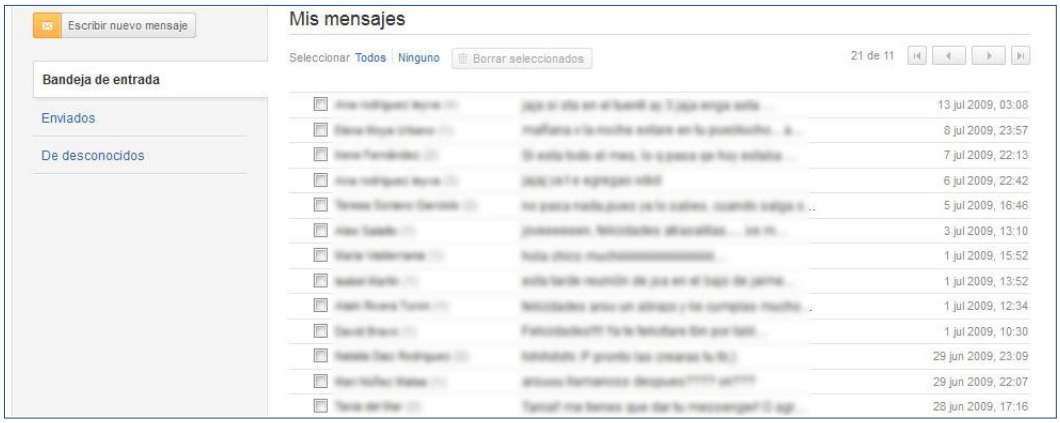


Figura 23 – Bandeja de entrada en Tuenti

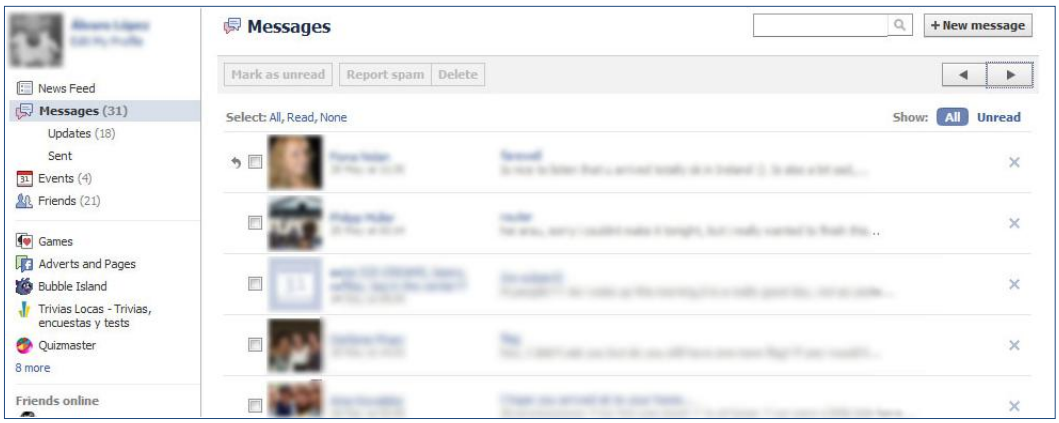


Figura 24 – Bandeja de entrada en Facebook

De las bandejas de entrada de Tuenti y Facebook destaco varias observaciones acerca de la información mostrada:

- **Título y texto del mensaje**: En Facebook se muestra el título y un pequeño resumen del texto de mensaje, en Tuenti se suprime la opción de título. De hecho, los mensajes en esta plataforma carecen de título.
- **Fotografía**: Mientras que en Facebook se muestra la fotografía de perfil del remitente, en Tuenti tan solo muestra su nombre.
- **Número de mensajes en la cadena**: En este caso es Tuenti quien muestra el número de mensajes ligados a una conversación (a la derecha del remitente), mientras que Facebook no hace nada similar.

En cuanto a la interfaz, también existen similitudes y diferencias:

- **Bandejas de entrada y salida, y nuevo mensaje**: La interfaz principal de la mensajería de ambas plataformas es la bandeja de entrada, desde la cual se tiene acceso a los mensajes enviados y a la opción de enviar un nuevo mensaje. En ambos casos la carga de una bandeja u otra se hacen sobre la misma página, evitando recargas en el navegador. Si bien a día de hoy la proliferación de este comportamiento es habitual, no está de más resaltarlo.
- **Paginación**: Ambas plataformas disponen de botones para avanzar y retroceder página en el listado de mensajes. En el caso de Tuenti el sistema de paginación es más completo pues retroalimenta al usuario indicándole en que página esta y cuantas páginas hay en total. También Tuenti permite avanzar en un solo clic hasta la primera o la última página. Curiosamente ninguna de las dos plataformas indica el número de mensajes en total.
- **Buscador**: Tuenti carece de buscador. Facebook dispone de uno que permite buscar tanto por usuario como por texto o título del mensaje en el mismo campo. Si bien no son objeto de estudio, servicios tan populares como GMail o Hotmail también disponen de un único campo para la realización de búsqueda.
- **Seleccionar mensajes/borrar mensajes**: Ambas plataformas permiten seleccionar directamente todos los mensajes. En el caso de Facebook se incluye además la posibilidad de seleccionar solo los mensajes leídos. A la hora de borrarlos se permite el borrado múltiple en ambas plataformas, pero tan solo Facebook habilita un botón de borrado individual para cada uno de los mensajes.

3.2.2 Visualización de mensajes

Si bien las bandejas de entradas eran similares en ambas plataformas, las diferencias existentes entre la visualización de mensajes en Tuenti y en Facebook es tan notable que merece la pena estudiarlas por separado. Estas diferencias se muestran en la Figura 25 y en la Figura 26.

Figura 25 - Visualización de mensajes en Tuenti

En Tuenti la visualización de una cadena de mensajes se realiza sobre la propia bandeja de entrada, desplegandose la cadena de mensajes al hacer clic sobre la fila correspondiente a la misma. Del mismo modo el cuadro para responder se muestra justo encima de los mensajes. Aparte de estas características no hay nada más destacable, salvo el hecho de poder adjuntar enlaces a fotos y videos de forma sencilla mediante el uso de los dos botones situados a la derecha del botón Enviar.

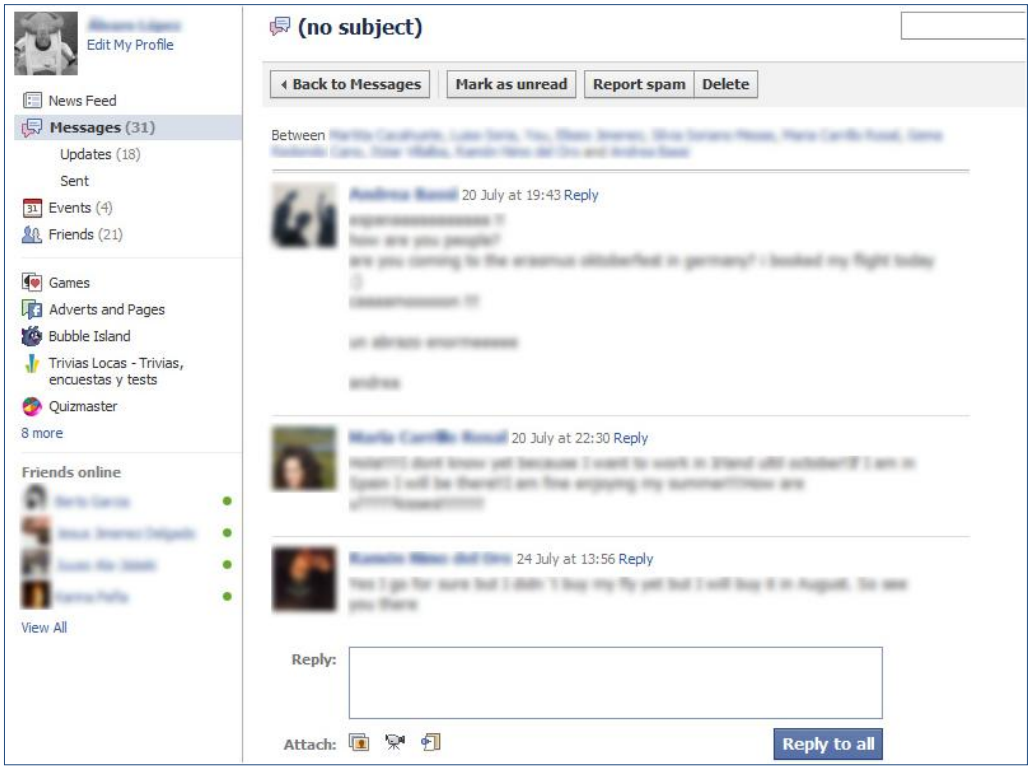


Figura 26 - Visualización de mensajes en Facebook

En Facebook en cambio la visualización de los mensajes tiene una interfaz independiente. En la parte superior de la interfaz aparece un elemento muy importante, los participantes en la cadena de mensajes. A diferencia de Tuenti, Facebook permite la mensajeria entre grupos de personas. Esta característica permite que varios usuarios puedan enviarse y recibir mensajes entre todos ellos sin tener que crear diversos canales de comunicación independientes. Del mismo modo a la derecha de cada participante de la conversación aparece un botón de respuesta individual, que genera una nueva conversación independiente tan solo con ese usuario, pero que mantiene un vínculo con la conversación de la que se independizo.

Esta característica resulta muy interesante para Tutor, puesto que muchas veces un profesor puede querer iniciar una cadena de mensajes con todo un grupo de alumnos, pero quizás despues de ese comienzo un alumno quiera iniciar una nueva cadena de mensajes independiente a raiz del tema inicial.

3.2.3 Nuevo mensaje

La Figura 27 y la Figura 28 muestran la interfaz de envío de mensajes en Tuenti y Facebook. Ambas plataformas comparten una interfaz muy similar, mostrando una pequeña ventana que se sobremuestra sobre el contenido que estamos visualizando en pantalla. En esta ventana escogemos destinatarios, título (solo en Facebook) y texto del mensaje. Una vez más apreciamos la diferencia resaltada en el punto 3.2.2, en el que comentabamos como Facebook permite el envio de un mismo mensaje a múltiples usuarios, creandose una comunicación con todos ellos.

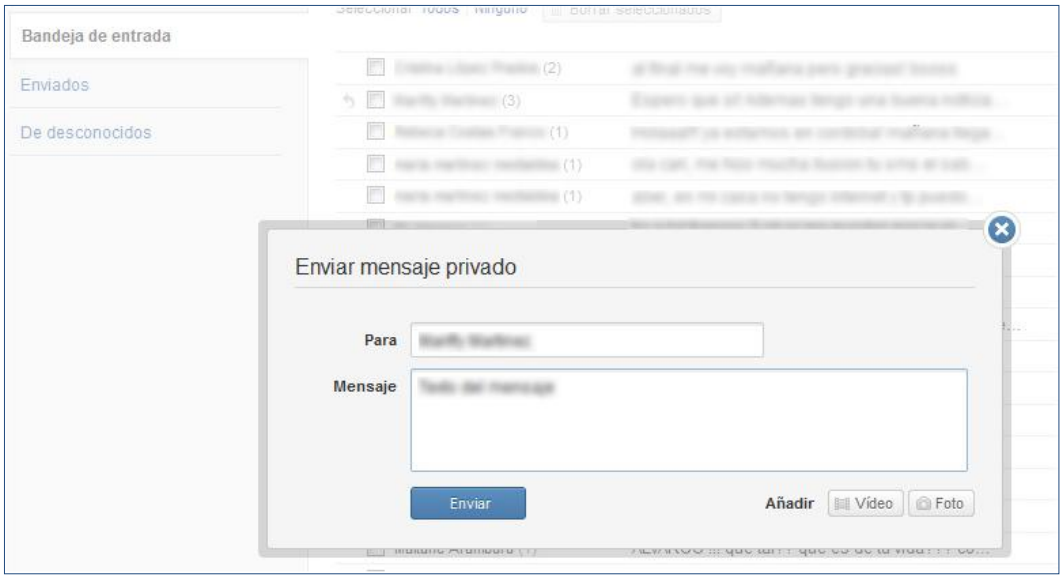


Figura 27 - Nuevo mensaje en Tuenti

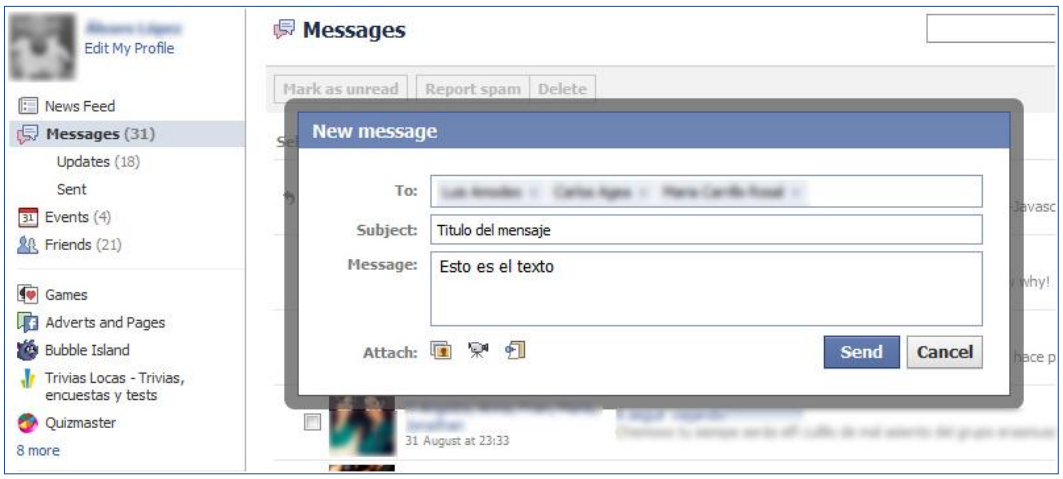


Figura 28 - Nuevo mensaje en Facebook

3.3. Especificación de requisitos

En los proyectos donde se aplica la ingeniería del software tradicional, obtener los requisitos de un sistema se suele hacer mediante entrevistas con el cliente y mediante el estudio de cómo trabaja el cliente con el sistema actual, en caso de que exista. En el caso de este módulo para Tutor la particularidad es que no existe cliente como tal, ni existe sistema previo. Por tanto yo mismo soy mi cliente y de mi depende especificar los requisitos que definan el funcionamiento del módulo a desarrollar. No obstante, dado el carácter iterativo del desarrollo de software, es de esperar que una vez desarrollado el módulo surjan nuevos requisitos a raíz del uso de este por parte de los usuarios.

3.3.1 Requisitos funcionales

Tras el estudio previo realizado a otras plataformas y un análisis de las funcionalidades de las que sería interesante dotar a Tutor en cuanto a mensajería, los requisitos funcionales que obtengo para el módulo son los siguientes:

- Envío de mensajes a uno o varios destinatarios del sistema. Para ello el usuario deberá estar identificado previamente en el sistema.
- Un usuario podrá enviar mensajes a otros usuarios que estén en la misma asignatura en las que él se encuentre. No habrá restricciones en cuanto al número de destinatarios, un usuario podrá enviar un mensaje a todos los profesores y grupos de alumnos de una asignatura.
- Posibilidad de búsqueda de cadenas de mensajes mediante título, texto o remitente de algún mensaje dentro de la cadena.
- En una cadena de mensajes con múltiples destinatarios un usuario podrá responder y leer los mensajes enviados por todos los receptores de la cadena. También podrá responder individualmente a un mensaje, generándose una nueva cadena de mensajes tan solo con el remitente de este mensaje.
- El usuario podrá borrar cadenas de mensajes completas o mensajes dentro de estas.
- El usuario tendrá diversos indicadores para conocer rápidamente cuando tiene mensajes nuevos, cuando un mensaje ya ha sido leído, y cuando ha sido respondido.
- Enlaces u otro tipo de información multimedia contenidas en un mensaje serán procesados por el sistema para mostrar la información pertinente en vez del texto plano.

3.3.2 Requisitos no funcionales

Los requisitos no funcionales vienen dados en gran parte por la plataforma en la que se integrará el modulo, Tutor:

- **Facilidad de uso**. Se respetarán las interfaces de usuario existentes en la plataforma y se homogeneizará el módulo con estas. Dado que muchos de los futuros usuarios del módulo probablemente estén familiarizados con los módulos de mensajería existentes en diversas plataformas de redes sociales, se intentará acercar en la medida de lo posible el funcionamiento del módulo a ejemplos ya existentes, con el objetivo de simplificar la curva de aprendizaje del usuario al nuevo módulo.
- **Rendimiento**. Dada las características de la plataforma, un sistema Web, se evitarán en la medida de lo posible las recargas de página, con el objetivo de agilizar la carga y visualización de mensajes. La potencia actual de los equipos hace que sea más rápido pedir tan solo los datos necesarios al servidor mientras se deja al cliente la responsabilidad de procesar y mostrar estos. En el apartado de almacenamiento de datos se evitará la redundancia de estos. Dado que un mensaje puede ser enviado a cientos de usuarios se debe buscar la forma de evitar el duplicado de estos mensajes en el sistema sin que ello implique una posible pérdida de datos.
- **Fiabilidad**. El sistema evitará los posibles errores que un módulo de estas características pueda producir, tales como mensajes vacios, borrados accidentales de datos. Se informará al usuario de los posibles errores de uso y funcionamiento, dándole la posibilidad de recuperarse de estos sin que se produzca una pérdida de datos.

3.3.3 Diagrama de casos de uso

La Figura 29 muestra el diagrama de los casos de uso existentes para la utilidad de la mensajería interna de Tutor. En el análisis de dicho diagrama se aprecia que las dependencias entre casos de uso vienen ligadas principalmente por lo que será la futura interfaz gráfica del usuario; es decir, casos de uso como *Eliminar una o varias cadenas de mensajes* o *Ver cadena de mensajes* no podrán ser realizados si no estamos previamente en casos de uso como *Consultar mensajes recibidos* o *Consultar mensajes enviados*, pues será a partir de éstos de donde se obtendrá un listado de mensajes.

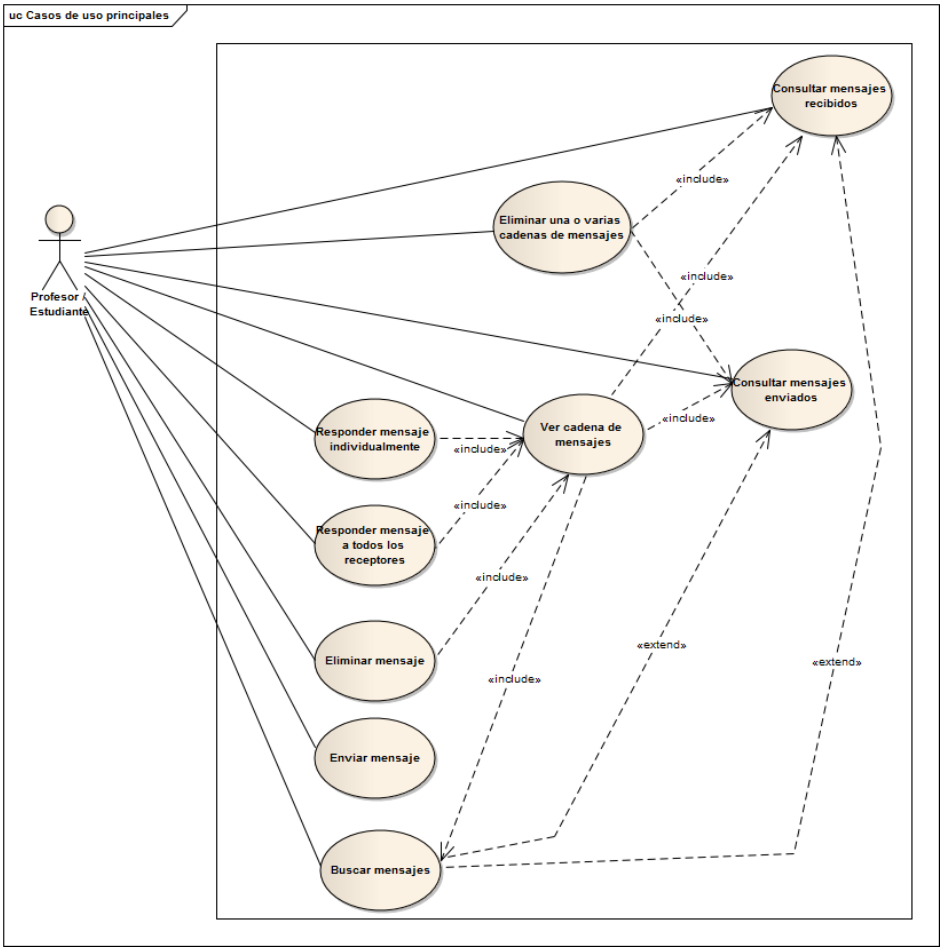


Figura 29 - Diagrama de casos de uso del módulo de Mensajería

3.3.4 Descripción de los casos de uso

Como se puede apreciar en la Figura 29, hay los siguientes nueve casos de usos, cada uno de los cuales se describirá a continuación en la correspondiente tabla:

1. Consultar mensajes recibidos
2. Consultar mensajes enviados
3. Ver cadena de mensajes
4. Enviar mensaje
5. Responder mensaje individualmente
6. Responder mensaje a todos los receptores
7. Eliminar mensaje
8. Eliminar una o varias cadenas de mensajes
9. Buscar mensajes

Caso de Uso	Consultar mensajes recibidos	CU-1
Actores	Profesor, Alumno	
Tipo	Primario, Esencial	

Dependencia					
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación.				
Postcondición					
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Consultar el listado de mensajes recibidos

Resumen
El usuario consulta el listado de mensajes recibidos

Curso Normal			
1	En el menú principal, el usuario acciona la opción *Mensajería interna*.	**2**	El sistema muestra la interfaz de la *Mensajería interna*, mostrando la lista de últimos mensajes recibidos y destacando los mensajes nuevos.

Cursos Alternos	
2.a	El usuario no tiene mensajes recibidos, por lo que se le muestra un mensaje indicando este hecho.
2.b	El usuario selecciona avanzar o retroceder página en la lista de mensajes recibidos, que se muestran de 10 en 10.

Comentarios
Desde la interfaz de *Mensajería interna* se puede acceder a la lista de mensajes recibidos en cualquier momento.

Caso de Uso	Consultar mensajes enviados				CU-2
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia					
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación.				
Postcondición					
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Consultar el listado de mensajes enviados

Resumen
El usuario consulta el listado de mensajes enviados

Curso Normal			
1	En el menú principal, el usuario acciona la opción *Mensajería interna*.	**2**	El sistema muestra la interfaz de la *Mensajería interna*, mostrando la lista de últimos mensajes recibidos y destacando los mensajes nuevos.
3	El usuario selecciona la pestaña de mensajes enviados.	**4**	El sistema muestra los últimos mensajes enviados.

Cursos Alternos	
2.a	El usuario no tiene mensajes enviados, por lo que se le muestra un mensaje indicando este hecho.
2.b	El usuario selecciona avanzar o retroceder página en la lista de mensajes enviados, que se muestran de 10 en 10.

Comentarios
Desde la interfaz de *Mensajería interna* se puede acceder a la lista de mensajes enviados en cualquier momento.

Caso de Uso	Ver cadena de mensajes				CU-3
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia	Consultar mensajes recibidos, Consultar mensajes enviados, Buscar mensajes.				
Precondición	El usuario está autenticado en Tutor y tiene permiso para ver la cadena de mensajes a la que está intentando acceder.				
Postcondición	La cadena de mensajes queda marcada como leída.				
Autor	Álvaro López Martínez	Fecha	1-09-2010	Versión	1.0

Propósito
Consultar una cadena de mensajes

Resumen
El usuario consulta una cadena de mensajes en particular

Curso Normal			
1	El usuario selecciona una cadena de mensajes de alguno de los listados obtenidos mediante los CUs Consultar mensajes recibidos, Consultar mensajes enviados o Buscar mensajes.	2	El sistema muestra la cadena de mensajes, y quiénes son los receptores de la misma.

Cursos Alternos	
1.a	El usuario también puede llegar a un mensaje mediante la URL directa de acceso al mismo. En este caso, se omite el listado previo.

Comentarios

Caso de Uso	Enviar mensaje				CU-4
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia					
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación.				
Postcondición	El mensaje queda guardado en el sistema y los destinatarios del mismo son notificados.				
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Enviar un mensaje

Resumen
El usuario envía un mensaje a uno o más usuarios del sistema

Curso Normal			
1	En el menú principal, el usuario acciona la opción *Mensajería interna*.	**2**	El sistema muestra la interfaz de la *Mensajería interna*, mostrando la lista de últimos mensajes recibidos y destacando los mensajes nuevos.
3	Desde la interfaz de *Mensajería interna* el usuario accede a la pestaña *Nuevo mensaje.*	**4**	El sistema muestra la interfaz de *Nuevo mensaje*, en la que aparece un listado de profesores, grupos y estudiantes de la asignatura que tiene activa el usuario.
5	El usuario selecciona los destinatarios, escribe el título y texto del mensaje, y presiona el botón de enviar mensaje.	**6**	El sistema crea una nueva cadena de mensajes, le asocia el mensaje enviado y notifica a los destinatarios del mismo.

Cursos Alternos	
4.a	El usuario no tiene ninguna asignatura. Se le muestra un mensaje informando de este hecho.
4.b	El usuario cambia de asignatura activa. Se actualiza la lista de profesores, grupos y estudiantes con respecto a la nueva asignatura.
5a	El usuario no selecciona ningún destinatario. Se le muestra un mensaje informando de este hecho y se le insta a que seleccione algún destinatario.
5.b	El usuario deja incompletos los campos de título o texto del mensaje. Se le muestra un mensaje informando de este hecho y se le insta a que rellene el campo incompleto.

Comentarios
El usuario podrá seleccionar uno o varios destinatarios para sus mensajes.

Caso de Uso	Responder mensaje individualmente				CU-5
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia	Ver cadena de mensajes				
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación. El usuario tiene algún mensaje.				
Postcondición	El mensaje queda guardado en el sistema y el destinatario del mismo es notificado.				
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Responder a un mensaje de forma individual

Resumen
El usuario responde a un mensaje de forma individual, independientemente de que el mensaje respondido fuera de una cadena entre 2 o entre más usuarios.

Curso Normal			
1	Desde el CU Ver cadena de mensajes el usuario acciona la opción "Responder sólo a esta persona" correspondiente a alguno de los autores de los mensajes existentes en la cadena.	**2**	El sistema muestra una nueva ventana en la que solicita el texto para el mensaje de respuesta.
3	El usuario completa el mensaje de respuesta y hace uso de la acción responder.	**4**	El sistema genera una nueva cadena de mensajes independiente de la original y añade a ésta el mensaje de respuesta, incluyendo como receptores al usuario y al autor del mensaje que se está respondiendo.

Cursos Alternos	
2.a	La cadena de mensajes ya era entre sólo dos usuarios. En este caso, el usuario responde desde un campo de texto existente a continuación de los mensajes de la cadena y NO se genera una cadena nueva.
3.a	Si el mensaje de respuesta está vacío, se muestra un mensaje informando de este hecho y se insta al usuario a rellenar el mensaje.

Comentarios

Caso de Uso	Responder mensaje a todos los receptores				CU-6
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia	Ver cadena de mensajes				
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación. El usuario tiene alguna cadena de mensaje con varios receptores.				
Postcondición	El mensaje queda guardado en el sistema y los destinatarios del mismo son notificados.				
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Responder a todos los receptores de una cadena de mensajes

Resumen
El usuario envía un mensaje a todos los receptores de una cadena de mensajes

Curso Normal			
1	Desde el CU Ver cadena de mensajes el usuario rellena el campo de texto situado al final de la cadena de mensaje, situado a continuación de todos los mensajes existentes en la cadena y hace uso de la acción responder.	**2**	El sistema vincula la respuesta a la cadena de mensajes y notifica a todos los receptores.

Cursos Alternos	
1.a	Si el mensaje de respuesta está vacío, se muestra un mensaje informando de este hecho y se insta al usuario a rellenar el mensaje.

Comentarios

Caso de Uso	Eliminar mensaje				CU-7
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia	Ver cadena de mensajes				
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación. El usuario tiene alguna cadena de mensajes.				
Postcondición	El mensaje queda eliminado del sistema para el usuario.				
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Eliminar un mensaje dentro de una cadena de mensajes

Resumen
El usuario elimina un mensaje dentro de una cadena de mensajes

Curso Normal			
1	Desde el CU Ver cadena de mensajes el usuario hace uso de la opción *Eliminar* sobre alguno de los mensajes existentes en la cadena.	**2**	El sistema elimina ese mensaje de la cadena de mensajes para el usuario activo.

Cursos Alternos	

Comentarios
Es importante recordar que una cadena de mensajes puede incluir varios destinatarios; por tanto, el hecho de que un destinatario borre un mensaje en particular no implica que el resto de destinatarios no puedan seguir viendo ese mensaje. Sólo el usuario que borra dicho mensaje pierde el acceso al mismo.

Caso de Uso	Eliminar una o varias cadenas de mensajes				CU-8
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia	Consultar mensajes enviados, Consultar mensajes recibidos				
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación. El usuario tiene alguna cadena de mensajes.				
Postcondición	La cadena de mensajes queda eliminada del sistema para el usuario.				
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Eliminar una o varias cadenas de mensaje

Resumen
El usuario elimina una o varias cadenas de mensajes

Curso Normal			
1	Desde el CU Consultar mensajes enviados o Consultar mensajes recibidos el usuario selecciona una o varias cadenas de mensajes y hace uso de la opción *Borrar las cadenas de mensajes seleccionadas*.	**2**	El sistema elimina la/s cadena/s de mensajes para el usuario activo.

Cursos Alternos	

Comentarios
Es importante recordar que una cadena de mensajes puede incluir varios destinatarios; por tanto, el hecho de que un destinatario borre una cadena de mensajes en particular no implica que el resto de destinatarios no puedan seguir viendo esa cadena de mensajes. Sólo el usuario que borra dicha cadena de mensajes pierde el acceso a la misma.

Caso de Uso	Buscar mensajes				CU-9
Actores	Profesor, Alumno				
Tipo	Primario, Esencial				
Dependencia	Consultar mensajes enviados, Consultar mensajes recibidos				
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación.				
Postcondición					
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Buscar un mensaje

Resumen
El usuario busca entre sus mensajes por el autor, título o texto del mismo

Curso Normal			
1	Desde el CU Consultar mensajes enviados o Consultar mensajes recibidos el usuario introduce el autor o texto que desea buscar en el cuadro de búsqueda y hace uso de la acción *Buscar*.	**2**	El sistema muestra un listado de los mensajes coincidentes con la cadena de texto buscada.

Cursos Alternos	
2.a	Si el sistema no encuentra ningún mensaje coincidente, lo indica mostrando un mensaje.

Comentarios

3.4. Análisis y diseño

3.4.1 Introducción

El principal problema que surgía a la hora de desarrollar este módulo era cómo mantener la consistencia y evitar la redundancia en el almacenamiento de los mensajes entre usuarios. Una primera aproximación al diseño del módulo era la de escoger una solución en la que por cada mensaje enviado se guardara una copia del mensaje, tanto para el emisor como para cada uno de los receptores. Así, si el remitente de un mensaje deseaba eliminar el mensaje de su bandeja de enviados, o si alguno de los receptores deseaba eliminar su mensaje de la bandeja de recibidos, tan sólo debería eliminarse la copia del mensaje creada para ese usuario. Esta aproximación estaría bastante cercana al funcionamiento de los sistemas de correo electrónico tradicionales, en el que cada destinatario tiene una copia del mismo en el servidor que aloja su cuenta. La Figura 15 muestra un ejemplo de esta idea.

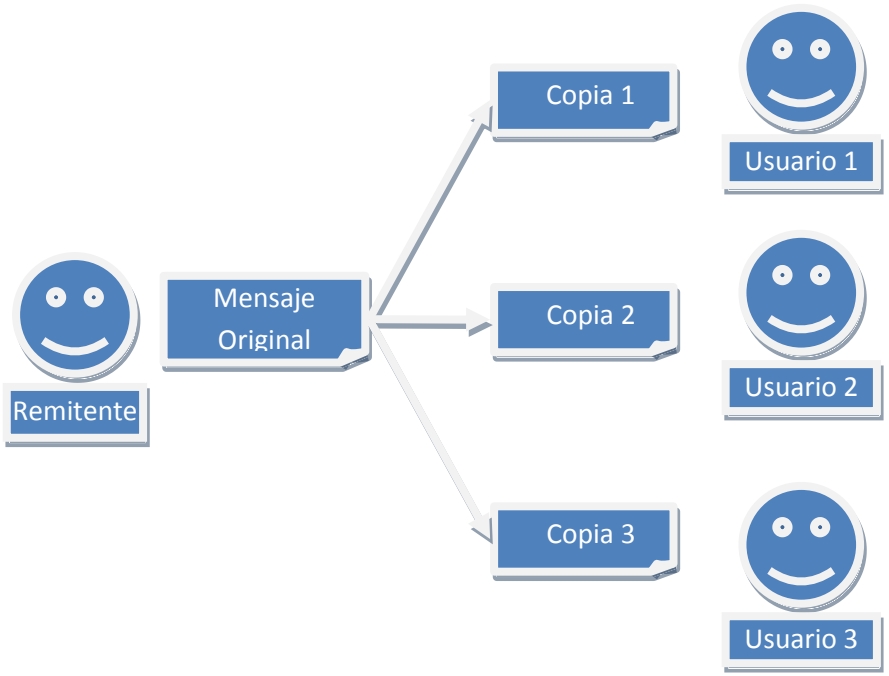


Figura 30 - Distribución de mensajes tradicional

Si bien esta forma de implementar el módulo era la más sencilla, también era la más ineficiente en cuanto a redundancia de los datos se refiere. Era bastante probable, aunque no se ha podido contrastar hasta su puesta en funcionamiento, que gran parte de los mensajes fuesen de un profesor para grupos completos de una asignatura. Es decir, aplicando esta primera aproximación se daría el caso de que un simple mensaje de un profesor a un grupo se duplicaría del orden de unas 80 veces, que es el tamaño medio que suele tener un grupo.

Dado que hay un único servidor para todos los usuarios de Tutor, a diferencia del correo electrónico tradicional, donde cada usuario puede usar servidores distintos, era más lógico guardar una sola copia del mensaje, y paralelamente establecer las relaciones necesarias para que cada uno de los receptores del mensaje pudiera acceder al mismo.

Teniendo en cuenta este hecho, se ha realizado una segunda aproximación en la que se guarda una única copia del mensaje y un estado del mensaje por cada receptor del mismo. Dada esta aproximación, se definen cuatro estados para la relación establecida entre un mensaje y un usuario. Los estados son los siguientes:

- **Mine (mío)**: Este estado se asigna a la relación establecida entre un mensaje y el usuario que lo ha creado.
- **Unread (no leído)**: Éste es el estado inicial para cada relación creada entre un determinado mensaje y cada uno de los receptores del mismo.
- **Read (leído)**: En el momento en que un usuario abre un mensaje, se pasa del estado unread al estado read.
- **Deleted (eliminado)**: Si el usuario elimina un mensaje, se anota de forma lógica en la relación entre el usuario y el mensaje, utilizando este estado.

Atendiendo a esta aproximación, se tiene una representación similar a la que se muestra en la Figura 31.

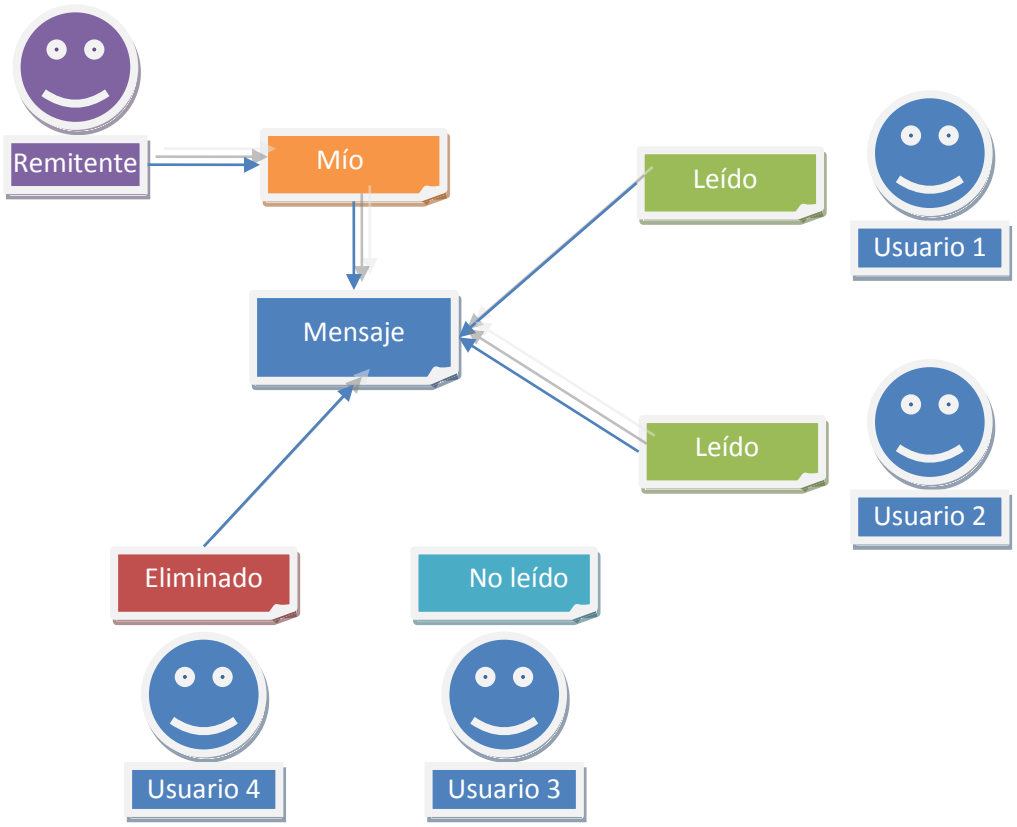


Figura 31 - Distribución de mensajes mediante estados

Otro problema de diseño que debía ser resuelto es el de cómo mantener los receptores de una cadena de mensajes. Uno de los requisitos funcionales era el de que un mensaje y sus respuestas pudiera ser enviado a múltiples destinatarios, y que los destinatarios fueran participes del resto de mensajes que se derivasen del mensaje inicial.

Una vez más se plantea el mismo problema de redundancia surgido a la hora de distribuir un mensaje. Dado un mensaje inicial, se deben almacenar todos los receptores del mismo, de cara a que las sucesivas respuestas también lleguen a todos los usuarios que eran destinatarios del mensaje inicial. Una primera aproximación para enviar estas respuestas podía consistir en consultar la lista de receptores del mensaje al que se está respondiendo, y enviar el mensaje a dichos receptores. De esta forma, cada vez que se envía un mensaje, se debe analizar el mensaje al que se está respondiendo y añadir los mismos receptores al mensaje de respuesta.

Sin embargo, resulta más eficiente crear el concepto de *cadenas de mensajes*. Se entiende como cadena de mensajes todos los mensajes vinculados a un mensaje inicial, teniendo estos mensajes como característica común el hecho de compartir los mismos receptores. Este concepto se puede ver en la Figura 32.

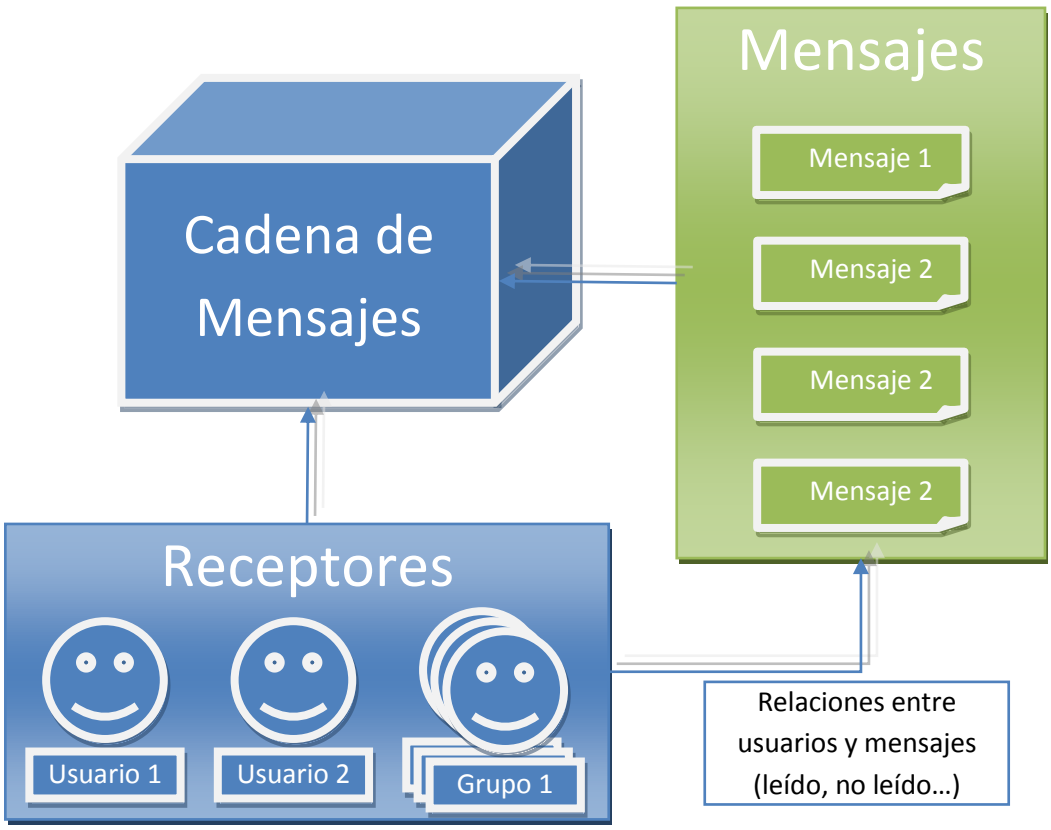


Figura 32 - Concepto de cadena de mensajes

Por último, partiendo de la base de que muchos mensajes se enviarán a un grupo completo, se da por hecho que un grupo puede ser receptor de una cadena de mensajes. Aplicando esta medida se consigue que, en vez de anotar los identificadores de cada uno de los miembros del grupo como receptores de una cadena de mensaje, se anote tan sólo el identificador del grupo al que pertenecen.

De esta forma, podemos tener cadenas donde los receptores sean varios alumnos, varios profesores, una combinación de varios grupos y varios profesores, varios grupos completos y varios alumnos de otros grupos, etc., minimizando siempre la cantidad de datos a almacenar.

En cualquier caso, sí que se deberá establecer una relación única entre cada uno de los receptores y los mensajes de una cadena. Es decir, aunque un receptor sea un grupo completo, se creará una relación para cada uno de los miembros del grupo.

3.4.2 Diagrama de clases

En el diagrama de clases de la Figura 33 se puede ver el diseño realizado a nivel de clases para el módulo de *Mensajería interna*. El principal elemento de este diagrama es la clase *PM_Thread*, la cual representa una cadena de mensajes. Esta clase se relaciona con la clase *PrivateMessage* y la clase *PM_Thread_Receivers*, que representan los mensajes vinculados a una cadena y los receptores de la cadena respectivamente. La clase *PM_Thread_Receivers* es una generalización de los tipos de receptores que puede tener una cadena, profesores, estudiantes y grupos, tal y como se explicó al final del apartado 3.4.1. Por último, la clase asociación *PrivateMessageStatus* representa la relación creada entre un usuario y un mensaje de una cadena.

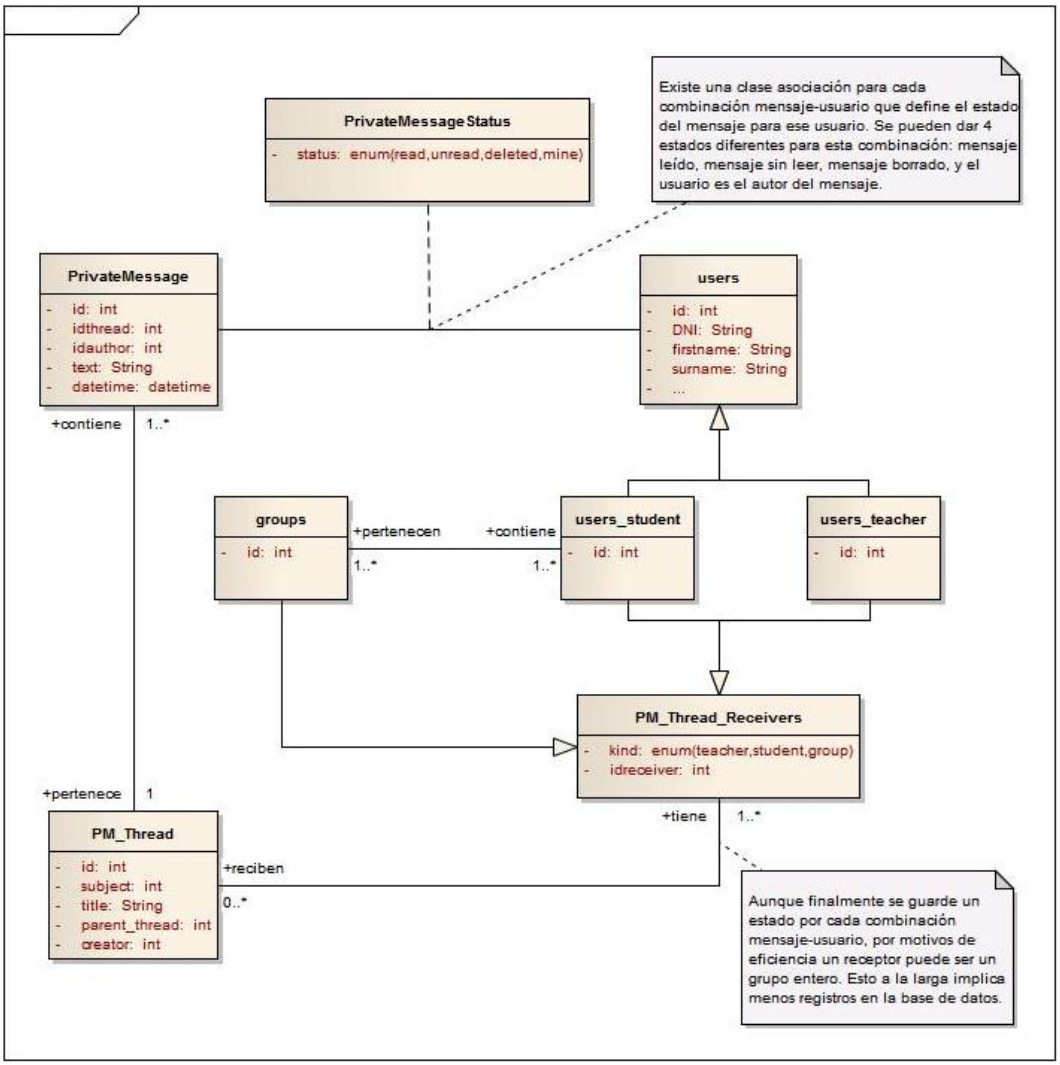


Figura 33 - Diagrama de clases del módulo de Mensajería Interna

3.5. Interfaz de usuario

A continuación se muestran una serie de capturas de pantalla del sistema de mensajería interna, que tratan de mostrar algunas pinceladas de su funcionamiento. Los mensajes mostrados son reales, pero en ningún caso son privados, ya que los mensajes mostrados son entre un alumno o un profesor con todos los alumnos o miembros de una asignatura.

En la Figura 34 y en la Figura 35 se muestra cómo se visualiza una cadena de mensajes con un único mensaje en ella. En la parte superior se muestra el título de la cadena de mensajes, así como la asignatura y el conjunto de usuarios a los que está asociada. A continuación, se muestra el único mensaje que hay en la cadena, incluyendo foto del remitente (que, en este caso, ha definido en las *Preferencias* de Tutor que no desea mostrar su foto al resto de usuarios), y la fecha y hora de envío del mensaje.

En la misma línea se presentan los iconos que dan la opción de responder individualmente al mensaje (es decir, sólo al emisor de ese mensaje concreto) y de eliminarlo respectivamente. Por último, en la parte inferior de la cadena de mensajes se muestra el campo de texto donde el usuario debe teclear su respuesta al mensaje y el botón *Responder a todos*, que enlaza dicha respuesta a la cadena de mensajes, enviándola a todos los receptores de la misma.

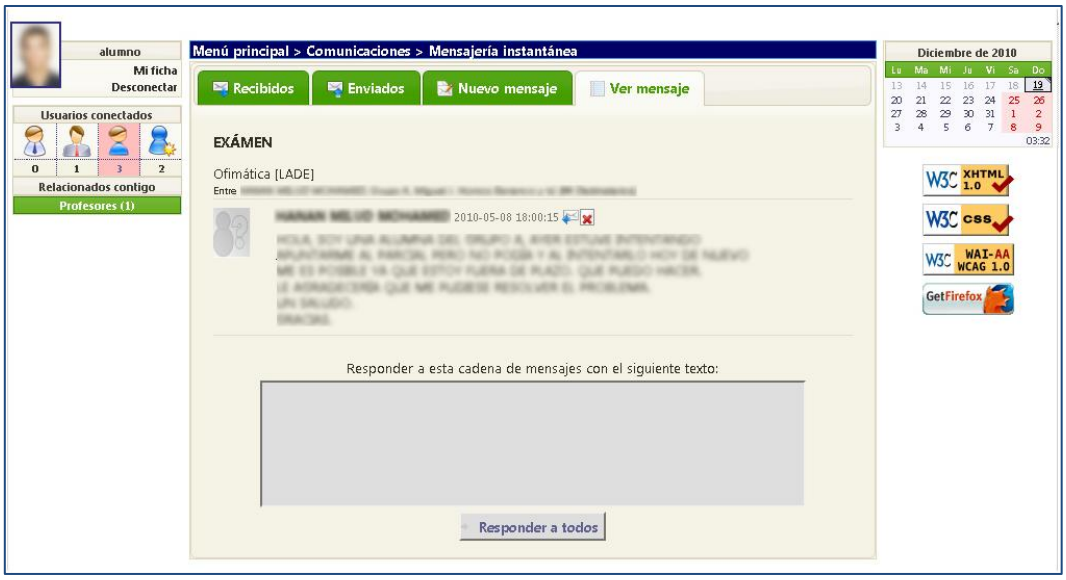


Figura 34 - Vista de un mensaje dentro de una cadena (estilo South Street)

La única diferencia existente entre la Figura 34 y la Figura 35 es el tema elegido en las *Preferencias* del usuario para mostrar la interfaz de usuario de la aplicación, presentándose ambas figuras únicamente para visualizar cómo este módulo de *Mensajeria interna* se adapta al tema escogido por el usuario en cada momento. Así, mientras que en la Figura 34 el tema usado es *South Street*, en la Figura 35 se utiliza el tema denominado *Redmond*.

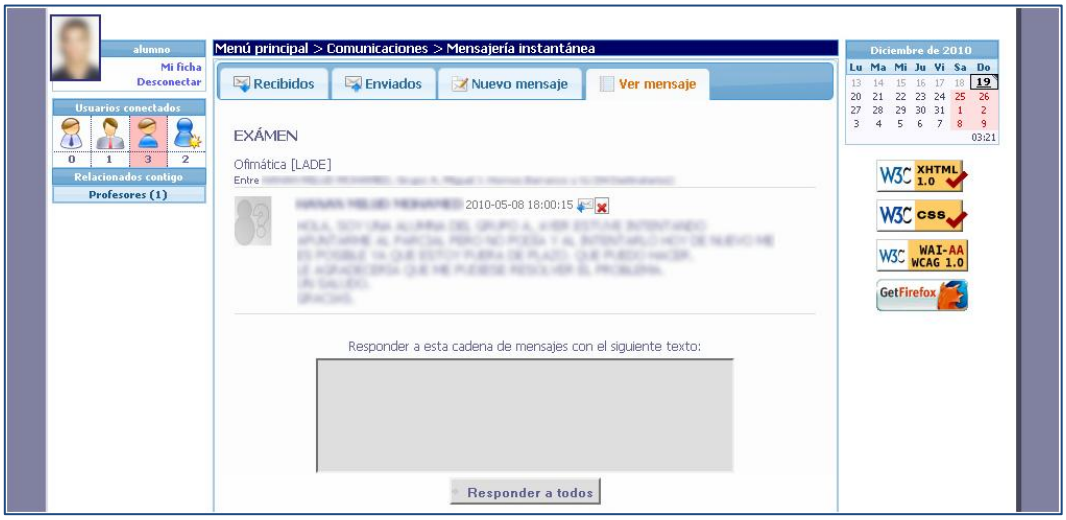


Figura 35 - Vista de un mensaje dentro de una cadena (estilo Redmond)

En la Figura 36 se puede ver una cadena de mensajes en la que participan múltiples usuarios. Al igual que en la Figura 34 y la Figura 35, cada mensaje lleva asociado el botón de responder individualmente y de eliminar mensaje.

Figura 36 - Vista de una cadena de mensajes entre múltiples usuarios

En la Figura 37 se muestra la vista de la *Bandeja de mensajes recibidos*. En la parte superior se encuentra el selector de asignatura con el que poder filtrar los mensajes por una asignatura u otra. A la derecha se muestra el campo que permite realizar una búsqueda entre los mensajes existentes y su botón asociado. A continuación, puede verse el contador de mensajes y los botones de navegación. Por último, se muestran cada uno de los mensajes de la página actual, con la foto y nombre del remitente, la fecha de envío, el título y un pequeño fragmento del texto del último mensaje recibido en la cadena.

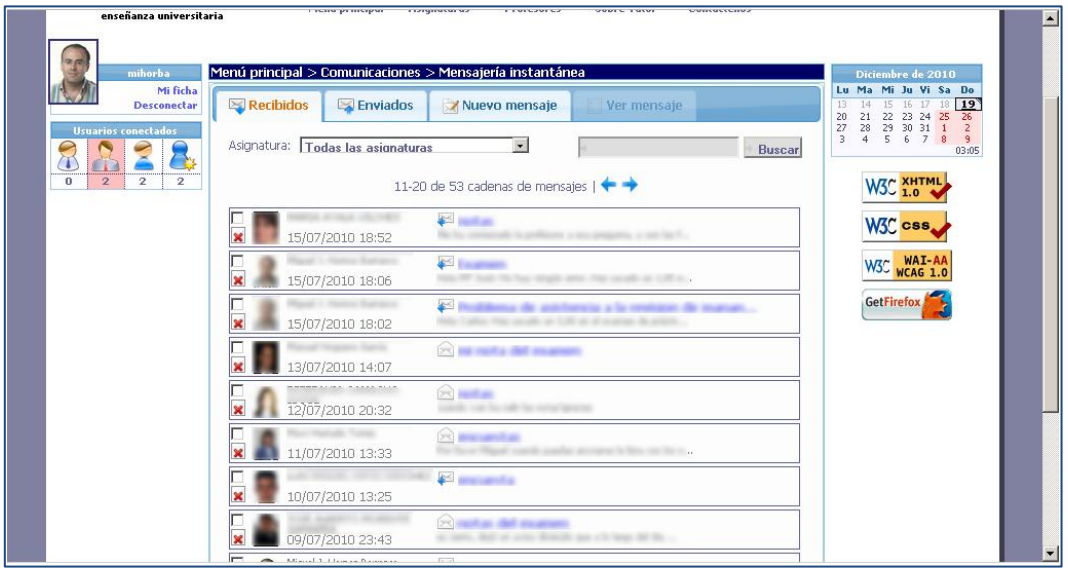


Figura 37 - Vista de la bandeja de mensajes recibidos

La Figura 38 muestra la interfaz para el envío de un nuevo mensaje. En la parte superior se permite elegir la asignatura y los destinatarios del mensaje, divididos en profesores, grupos de teoría y prácticas, y estudiantes de la asignatura.

Al seleccionar un destinatario, éste queda marcado con un suave sombreado en rojo. Si el destinatario marcado fuera un grupo automáticamente se marcarían todos los estudiantes del grupo en el recuadro de estudiantes.

A continuación de los destinatarios se muestran los campos de título (*Asunto*) y texto del mensaje, junto al botón que permite realizar el envío.

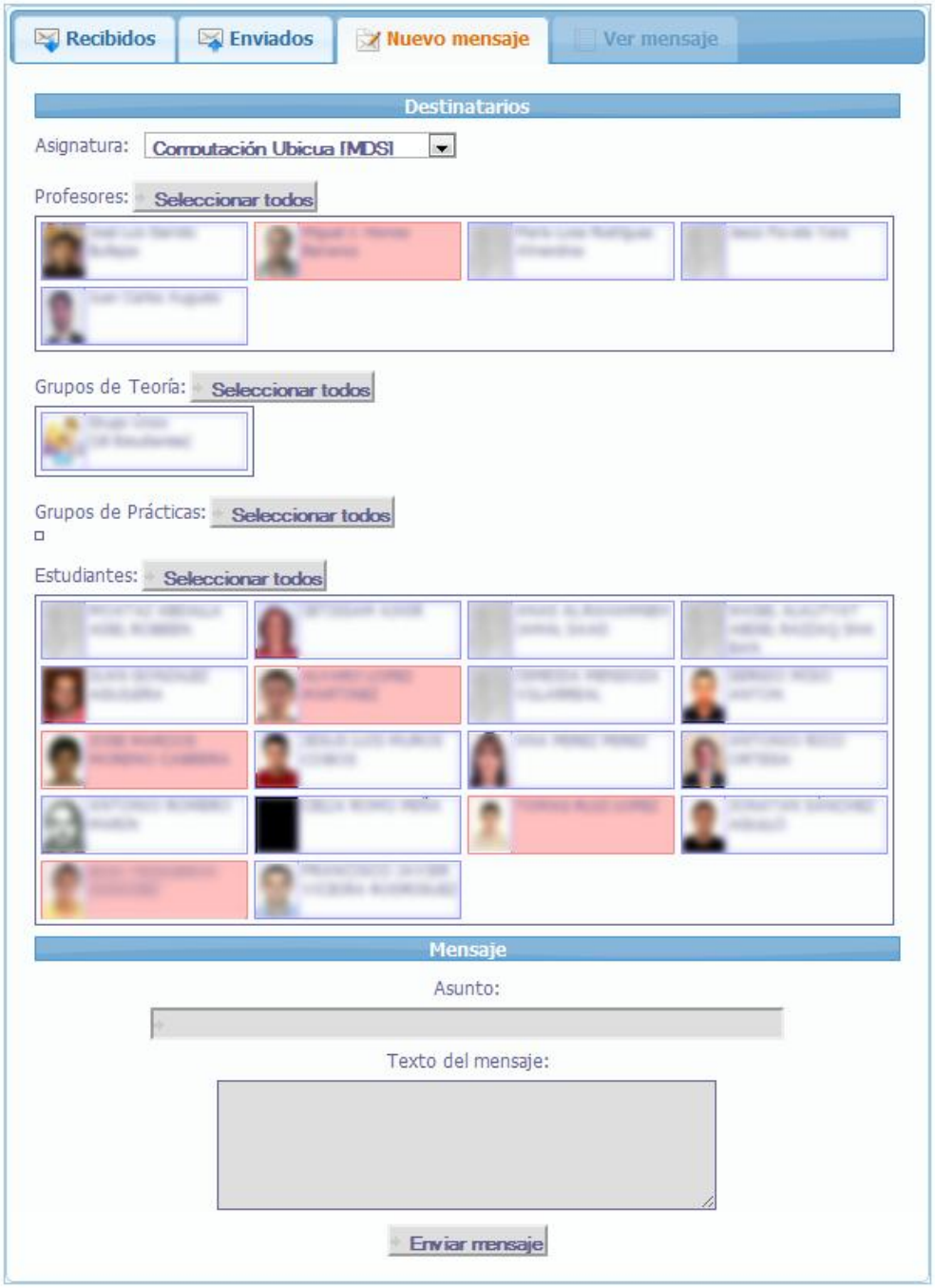


Figura 38 – Vista de la ventana de envío de nuevo mensaje

Capítulo 4. Chat

4.1. Motivación

Para explicar la motivación de la inclusión de un sistema de chat en Tutor, una vez desarrollado un módulo de mensajería interna, hay que analizar las ventajas que supone este módulo frente al módulo desarrollado en el Capítulo 3.

La principal ventaja es que un sistema de chat permite una comunicación totalmente síncrona, para que se produzca esta comunicación, se debe dar la situación de que tanto emisor y receptor estén conectados al sistema. A diferencia del módulo de mensajería interna, mediante un chat, el mensaje es recibido de forma instantánea por el receptor.

Si se hace un poco de repaso a la historia de Internet en España, se puede analizar cómo han evolucionado los sistemas de mensajería. Hace 12 años, la mayoría de las comunicaciones se realizaban mediante un software conocido como ICQ [18] o mediante las redes conocidas como IRC (Internet Relay Chat) [25]. Desde entonces hasta hoy, las comunicaciones, como si de modas se tratasen, han pasado por varias épocas. Si bien siguen existiendo redes de IRC y se puede seguir usando ICQ, a día de hoy los medios de comunicación más usados en la actualidad suelen ser Microsoft Windows Live Messenger [34], o Skype [46].

En 2008, Facebook amplió sus funcionalidades, añadiendo un chat integrado en la propia plataforma web. Un artículo de 2008 de la revista Mashable [47] recoge el siguiente comentario:

> **Facebook chat maybe won't replace your other chat applications, but it'll do just fine for chatting with people who you usually don't chat with.**

(El chat de Facebook quizás no reemplace a tus otras aplicaciones de chat, pero proporciona lo justo para chatear con gente con la que usualmente no sueles hacerlo)

El hacer referencia a esta cita no es casual, pues probablemente muchos de los usuarios de Tutor sólo sean compañeros de clase y ni mucho menos chateen entre ellos. Pero esto no significa que si estos compañeros de clase tienen una herramienta con la que poder comunicarse no vayan a hacer uso de ella. En el caso de Facebook, una nota [8] publicada en junio del 2009 por uno de sus ingenieros, Chris Piro, anunciaba que, un año después de la puesta en funcionamiento del servicio de chat de Facebook, éste había alcanzado la cifra de 1 billón de mensajes enviados por día.

4.2. Estudio de soluciones existentes

Los sistemas de chat se pueden clasificar en dos grandes grupos atendiendo a las dos principales formas de implementarlos. El principal objeto de estudio es el modo en el que se implementa el *back-end* del sistema. A este respecto se puede distinguir entre dos categorías principales, las soluciones basadas en el protocolo HTTP/AJAX, que hace uso de un servidor web, como Apache, o las soluciones basadas en otros protocolos, como el protocolo XMPP/Jabber. Ambas opciones se analizarán en los siguientes subapartados.

4.2.1 Soluciones basadas en HTTP/AJAX

En este tipo de soluciones se suele hacer uso de sistemas Apache+MySQL, tal y como Tutor está desarrollado hasta la fecha. El principal problema de este tipo de soluciones es que el cliente necesita hacer peticiones constantemente sobre el servidor Apache para determinar si tiene mensajes nuevos. Y Apache no está pensado para esta tarea. Otra posibilidad, haciendo también uso de Apache, consiste en mantener conexiones activas. Pero, del mismo modo, Apache no está pensado para mantener conexiones activas en el tiempo, pues en general las peticiones HTTP suelen consistir en una petición GET o POST a la que se le da una respuesta, después de la cual se cierra la conexión.

En un sistema como Tutor, que en la actualidad alcanza picos de hasta 100 usuarios simultáneos y que, en un futuro, podría ampliar esta cifra, se hace inviable pensar en un sistema basado en un servidor Apache.

4.2.2 Soluciones basadas en XMPP/Jabber

Afortunadamente existen soluciones basadas en otros protocolos. Un análisis mediante la herramienta Firebug [10] a la red social Tuenti, que se puede ver en la Figura 39 muestra cómo esta red social hace uso del protocolo XMPP/Jabber en la funcionalidad de chat que integró en septiembre del 2009 [48].

Otra herramienta conocida que también hace uso del protocolo XMPP/Jabber es Google Talk [15]. Si bien Google Talk es una aplicación de mensajería, en el servicio de correo Google Mail también se hace uso del protocolo XMPP/Jabber para integrar un sistema de chat en la plataforma web.

Dadas estas dos importantes referencias, y siendo XMPP/Jabber un protocolo abierto, a la hora de integrar un sistema de chat en Tutor se buscará una solución que esté basada en este protocolo, explicado en el punto 4.3. Para dicha solución se necesita un servidor y un cliente que puedan ser adaptados y utilizados en Tutor de forma fácil.

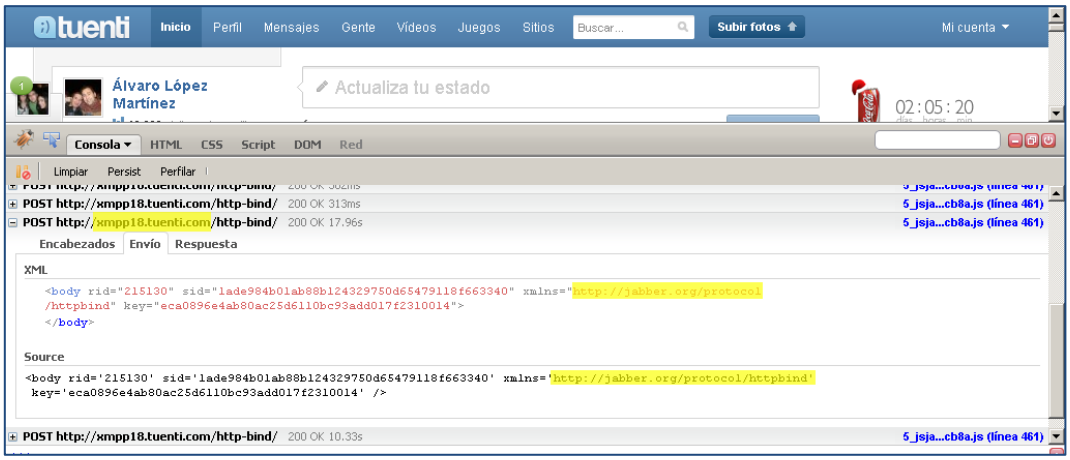


Figura 39 - Uso de XMPP/Jabber en Tuenti

4.3. El protocolo XMPP/Jabber

El protocolo Jabber comenzó a desarrollarse en 1998 por Jeremie Miller, siendo liberada su primera versión en Mayo del 2000. Este protocolo sentó las bases para el protocolo XMPP. Las principales características y las ventajas de cara a usarlo en Tutor son las siguientes:

Estándar abierto: La *Internet Engineering Task Force* (IETF) [24] define las especificaciones del protocolo en el RFC3920 [42] y RFC3921 [44]. El protocolo no está ligado a ninguna empresa y no hay que pagar por su uso. Por tanto, no existe problema alguno a la hora de usarlo en Tutor.

Diversas implementaciones: Según la *XMPP Standards Foundation* [56], existen 23 servidores en la actualidad que implementan el protocolo XMPP. Según esta misma página, existen 80 implementaciones de clientes. Esta amplia oferta facilita la elección de una solución idónea para Tutor.

Seguridad: Los servidores XMPP poseen sistemas de seguridad, tales como SASL y TLS.

Flexibilidad: Se pueden implementar funcionalidades a medida sobre XMPP. Aunque en principio no se pretende ampliar las funcionalidades que proporciona el propio protocolo, es una característica interesante de cara al futuro. Por otra parte, la página de *XMMP Standards Foundation* registra una larga lista de extensiones existentes [55].

4.4. El cliente iJab

A pesar de la cantidad de clientes XMPP existentes, encontrar uno que fuera adecuado para ser usado en Tutor no ha sido sencillo. De hecho, a día de hoy, la adaptación del cliente utilizado sigue siendo fuente de problemas, por motivos que se estudiarán más adelante.

En primer lugar, el cliente buscado debía poder ser integrado fácilmente en Tutor, en cuanto a interfaz se refiere. A diferencia del módulo de *Mensajería Interna*, explicado en el Capítulo 3, en este caso no se trataba de una funcionalidad a la que el usuario debe acceder expresamente pulsando una opción de menú, sino de una funcionalidad que debía estar presente mientras el usuario estuviese conectado al sistema. Para entender mejor esta idea, se puede observar la Figura 40, donde se muestra un *Muro* de Facebook. En la parte baja aparece la *barra de chat*. Esta barra permanece visible a lo largo de toda la navegación en Facebook, mostrando los contactos conectados en ese momento y permitiendo mantener una conversación con ellos independientemente de la sección de la red social en la que nos encontremos.

Por otra parte, era recomendable, aunque no imprescindible, que el cliente buscado no requiriese la instalación de plugins o complementos adicionales en el navegador.

Por último, encontrar un cliente que estuviese adaptado a la interfaz de usuario jQueryUI, descrita en el apartado 2.5.9 de esta memoria (página 29), facilitaba su integración en Tutor.

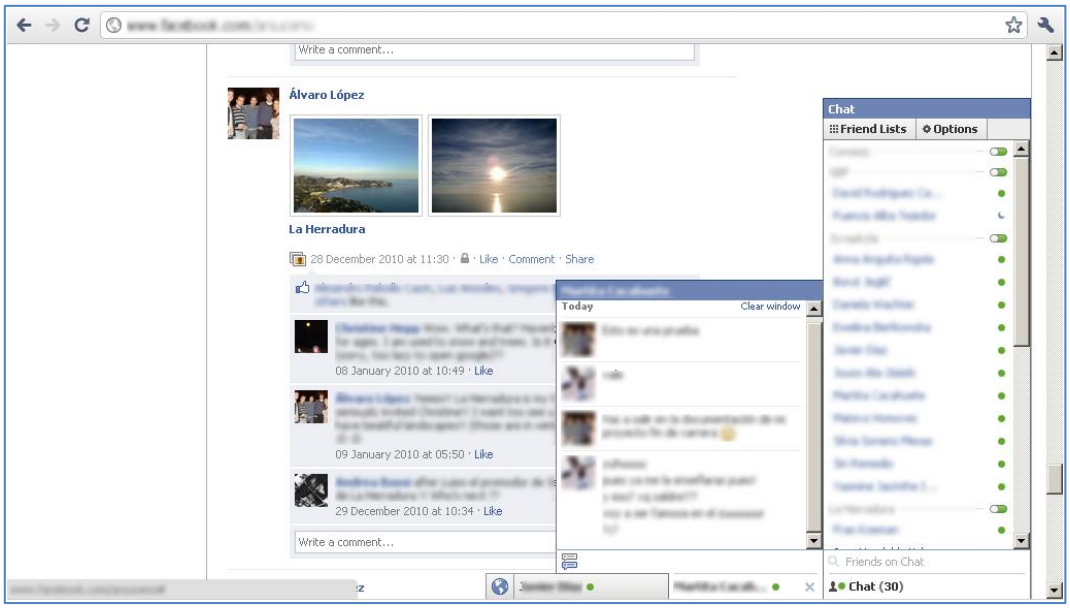


Figura 40 – Barra de chat en Facebook

Tras una larga búsqueda, el cliente elegido fue iJab [22]. Este cliente reúne las siguientes características:

Es código abierto

iJab está basado en otros proyectos de código abierto, como las Google *Webmaster Toolkits* (GWT) [17] o las librerías XMPP para GWT (XMPP4GWT) [57]. Por tanto, puede ser modificado y adaptado a los requisitos de Tutor.

Desarrollado en GWT

iJab está escrito íntegramente en Java, haciendo uso de las *Google Webmaster Toolkit*. Este proyecto de Google permite crear aplicaciones AJAX en Java, compiladas posteriormente en código Javascript. Para la adaptación del cliente iJab a Tutor, he tenido que familiarizarme con GWT. Esto se describirá con más detalle en el apartado 4.6, Adaptación a Tutor.

Tres tipos de interfaz

iJab se presenta en tres tipos de interfaz, siendo una de ellas la interfaz de *barra de chat* descrita anteriormente, y mostradas en la Figura 40 y en la Figura 41.

Figura 41 - Interfaz del cliente iJab en modo barra de chat

Interfaz amigable y compatible con jQueryUI

La interfaz de iJab es bastante amigable y está adaptada a los temas de jQueryUI; por tanto, su uso en Tutor se adaptará al estilo del sistema. Adicionalmente, soporta emoticonos y sonidos.

Basado en el estándar XMPP

iJab garantiza compatibilidad con el estándar XMPP. Adicionalmente, se recomienda su uso con dos de los servidores XMPP más utilizados, ejabberd [6] y Openfire [38]. Esta recomendación será una de las claves en la elección del servidor, proceso descrito en el apartado 4.5.

4.5. El servidor Openfire

El servidor elegido para ser adaptado a Tutor ha sido el servidor Openfire. Esta elección responde a una serie de criterios, que son los siguientes:

Es código abierto

Al igual que el cliente iJab, Openfire en un proyecto de código abierto, bajo licencia GNU. Esto permite hacer las modificaciones necesarias para su integración con Tutor. Por otra parte, el servidor continúa en desarrollo, habiéndose lanzado la última versión del mismo hace menos de 3 meses. Esto garantiza que en el futuro se seguirán recibiendo actualizaciones para los distintos fallos de seguridad que puedan surgir. Además tiene versión para Windows (Figura 42), sistema operativo usado actualmente en el servidor de Tutor.

Figura 42 - Consola del servidor Openfire

Amplia comunidad y documentación

Openfire cuenta con una amplia comunidad de desarrolladores y dispone de abundante documentación, así como guías para la integración del mismo en bases de datos existentes. Dado que uno de los objetivos finales era que el servidor instalado usara la base de datos de Tutor, éste ha sido un criterio clave en la elección de Openfire.

Escrito en Java

Openfire está escrito en su totalidad en Java. Éste es un criterio meramente personal, puesto que Java es uno de los lenguajes a los que estoy más habituado.

Compatible con iJab

La propia documentación de iJab explica su integración con Openfire y recomienda el uso de este servidor.

4.6. Adaptación a Tutor

La integración de Openfire e iJab en Tutor ha sido, cuanto menos, difícil. Pensar que, por ser Openfire e iJab soluciones ya desarrolladas, implica que hacerlas funcionar en Tutor es trivial es un pensamiento bastante alejado a la realidad. De hecho, pese a que existe una versión funcional de Openfire e iJab integrados en Tutor, como se puede ver en la Figura 43, aún a día de hoy existen problemas que deben ser resueltos para poder migrar el chat a la versión en producción de Tutor.

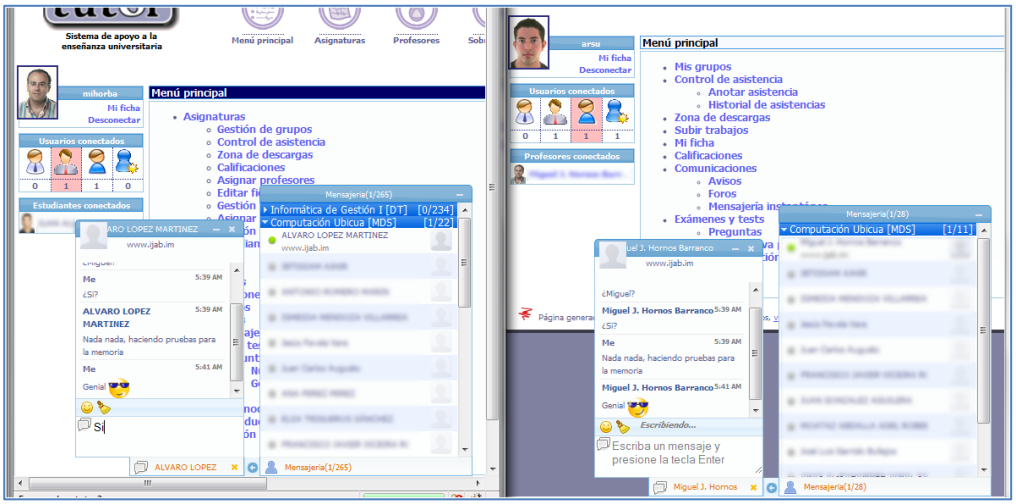


Figura 43 - Integración de Openfire e iJab en Tutor

A continuación se explican los diversos problemas resueltos en la integración y cómo se han solucionado. Más adelante, en el apartado 0, se explican los problemas que quedan por resolver.

4.6.1 Sincronización de usuarios de Tutor con Openfire

Para entender este punto, hay que profundizar en las características del protocolo XMPP. En la Figura 44 se puede ver el flujo típico de una comunicación XMPP. Ésta se divide en 5 pasos [11]:

1. El actor 1 inicia una conexión con el servidor.
2. El actor 1 recupera su lista de contactos (roster).
3. El actor 1 envía su estado (p.ej. Conectado) al servidor, que lo difunde a los usuarios de su lista de contactos.
4. El actor 1 recibe una lista de contactos conectados.
5. El actor 1 inicia la un intercambio de mensajes con otro actor.

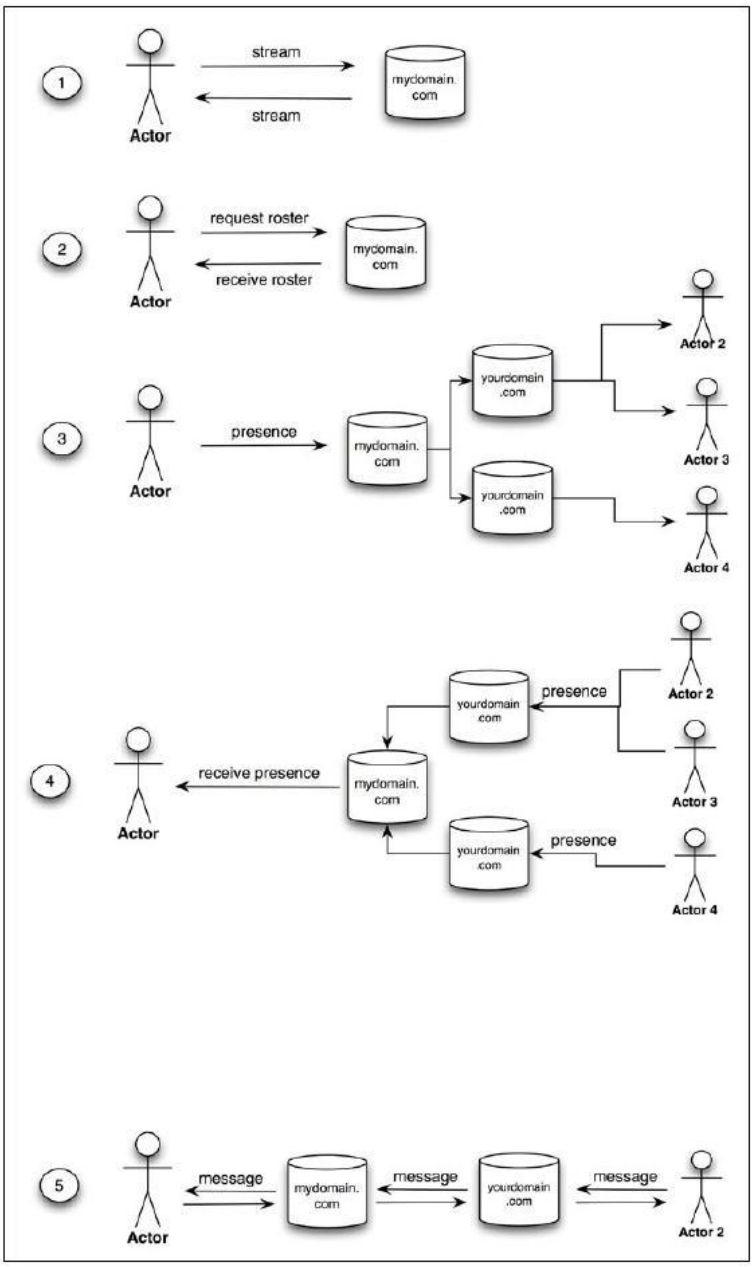


Figura 44 - Flujo de una comunicación XMPP (Fuente: [11])

En este flujo de comunicación aparece el concepto de *lista de contactos*. Para la integración con Tutor, se ha supuesto que la lista de contactos de un usuario son todos aquellos profesores y alumnos que comparten asignatura con dicho usuario.

Para reflejar este hecho en el servidor Openfire, se ha hecho uso de las posibilidades de integración que brinda. No obstante, dadas las características de existencia de varios roles en Tutor (administrador, profesor, estudiante) y que en un principio ni el protocolo XMPP ni el servidor Openfire distinguen roles de usuario, se han tenido que ampliar las posibilidades que brindaba el servidor. Por otra parte, XMPP contempla grupos de usuarios, el problema surge en cómo indicar a Openfire que son todos los estudiantes y profesores de una asignatura los que conforman un grupo.

Para entender esto, hay que explicar cuáles son las posibilidades de integración de Openfire. El servidor permite usar una base de datos existente y definir una serie de parámetros, concretamente sentencias SQL, con las que poder obtener los datos de una estructura de base de datos existente. Las consultas a definir son las siguientes:

- **jdbcAuthProvider.passwordSQL**: obtiene la contraseña de un usuario.
- **jdbcUserProvider.loadUserSQL**: obtiene el nombre y el correo electrónico de un usuario dado.
- **jdbcUserProvider.userCountSQL**: determina el número total de usuarios.
- **jdbcUserProvider.allUsersSQL**: recupera todos los nombres de usuario.
- **jdbcUserProvider.searchSQL**: realiza búsquedas de usuarios, mediante el nombre o el email de los mismos.

Aparte de estas consultas también hay que definir una serie de valores:

- **usernameField**: nombre del campo que almacena el nombre de usuario en la base de datos.
- **nameField**: nombre del campo que almacena el nombre real de un usuario en la base de datos.
- **emailField**: nombre del campo que almacena el email de un usuario en la base de datos.
- **jdbcAuthProvider.passwordType**: formato en el que está almacenado la contraseña, a escoger entre texto plano, encriptación mediante el algoritmo MD5 o encriptación mediante el algoritmo SHA-1. En el caso de Tutor, las contraseñas están almacenadas en formato SHA-1.

Adicionalmente a las consultas relacionadas con los usuarios, también se permite definir una serie de consultas relacionadas con los grupos:

- **jdbcGroupProvider.groupCountSQL**: determina el número total de grupos. En el caso de Tutor, se corresponde con la consulta para determinar el número total de asignaturas.

- **jdbcGroupProvider.allGroupsSQL**: recupera el nombre de todos los grupos de usuarios. En el caso de Tutor, cada nombre de grupo corresponderá con el nombre de la asignatura, seguido del acrónimo de la titulación entre corchetes (p.ej. "Computación Ubicua [MDS]").
- **jdbcGroupProvider.userGroupsSQL**: recupera todos los grupos de un determinado usuario. En el caso de Tutor, serán todas las asignaturas a las que un profesor esté asociado o todas las asignaturas donde un estudiante esté matriculado. No se deben confundir grupos de Openfire con grupos de asignaturas en Tutor. En el caso de Openfire un grupo es cada una de las listas de contactos que puede tener un usuario. Haciendo un símil a una herramienta de chat tradicional, podríamos tener listas de contactos tales como "Amigos", "Familiares", "Compañeros de trabajo", etc.
- **jdbcGroupProvider.descriptionSQL**: recupera la descripción de un grupo. En el caso de Tutor, este valor no es necesario.
- **jdbcGroupProvider.loadMembersSQL**: recupera todos los miembros de un grupo. En el caso de Tutor, será la unión entre los estudiantes y profesores relacionados con una asignatura.
- **jdbcGroupProvider.loadAdminsSQL**: recupera todos los administradores de un grupo. En el caso de Tutor, este valor no es necesario.

Llegado a este punto, surge un problema que las posibilidades de integración de Openfire no permiten resolver. En Tutor, la forma de recuperar las asignaturas relacionadas con un determinado profesor y aquéllas relacionadas con un alumno en concreto es totalmente distinta. Mientras para un profesor existe una relación directa entre profesor y asignatura en la base de datos, para un alumno la relación es con uno o más grupos de la asignatura en cuestión. La diferencia es tal que resulta insalvable si no se realizan modificaciones en el código fuente de Openfire.

Para ello, se hace imprescindible estudiar la clase *jdbcGroupProvider* de Openfire. Esta clase implementa, al igual que las clases que se pueden ver en la Figura 45, la interfaz definida por *GroupProvider*.

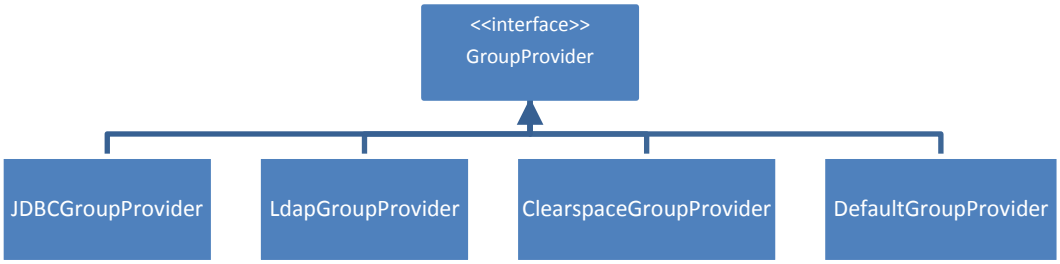


Figura 45 - Extracto del diagrama de clases de Openfire

La interfaz definida por *GroupProvider* define, entre otras, la función *Collection<String> getGroupNames(JID user)*. Esta función recibe el identificador de un usuario y devuelve la colección de grupos a los que pertenece.

Por tanto, se debe modificar esta función para que recoja los grupos en función de si el usuario es profesor o alumno. Una primera aproximación sería comprobar desde la función si el identificador del usuario pertenece a un profesor o a un estudiante y, en función de ello, ejecutar una sentencia u otra, pues las consultas son diferentes para un tipo de usuario u otro. Dado que el servidor Openfire no conoce el rol desempeñado por el usuario autenticado, se debería hacer una consulta previa a la base de datos para determinarlo, y en función de ello, realizar una consulta u otra.

Sin embargo, resulta bastante más eficaz ejecutar las consultas correspondientes, tanto para el profesor como para el alumno, y devolver la unión de las salidas de éstas. En ambos casos, se requiere realizar dos consultas: las correspondientes a la comprobación del tipo de usuario y a la carga de sus grupos en la primera aproximación, y las relativas a la carga de grupos de un profesor más la de la carga de grupos de un estudiante en la segunda aproximación. Además, con esta segunda aproximación se anticipa el sistema a un posible futuro en el que un profesor pudiese también ejercer el rol de estudiante, y viceversa. Es por ello que se ha optado por esta segunda solución.

Para mantener el criterio en el código y no "incrustar" las consultas en el mismo, se han añadido dos parámetros nuevos:

- **jdbcGroupProvider.userStudentGroupsSQL**: para recuperar los grupos de un alumno. Como ya se dijo antes, de cara a Openfire estos grupos son las diferentes asignaturas en las que está matriculado el alumno.
- **jdbcGroupProvider.userTeacherGroupsSQL**: para recuperar los grupos de un profesor. Al igual que en el caso del alumno, los grupos de Openfire de un profesor serán las asignaturas a las que esté vinculado en Tutor.

Por otra parte, para determinar los miembros de un grupo, también se han añadido dos nuevos parámetros:

- **jdbcGroupProvider.loadMembersStudentSQL:** para recuperar la lista de identificadores de alumnos en una asignatura de Tutor.
- **jdbcGroupProvider.loadMembersTeacherSQL**: para recuperar la lista de identificadores de profesores vinculados a una asignatura en Tutor.

Por último, de cara a determinar el año académico actual a usar en las consultas, se ha añadido el parámetro **jdbcGroupProvider.academicYear**.

Mediante estos parámetros, y tras la modificación de las correspondientes funciones en Openfire, se permite que éste pueda recuperar la lista de asignaturas, usuarios, grupos de usuarios y demás datos de la base de datos de Tutor. El valor de cada uno de los parámetros tras la integración a Tutor se muestra en la Figura 46.

Propiedad	Valor
jdbcUserProvider.usernameField	id
jdbcUserProvider.userCountSQL	SELECT COUNT(*) FROM sysusers
jdbcUserProvider.searchSQL	SELECT id FROM sysusers WHERE
jdbcUserProvider.nameField	login
jdbcUserProvider.loadUserSQL	SELECT CONCAT(firstname,' ',surname), email FROM users WHERE id=?
jdbcUserProvider.emailField	email
jdbcUserProvider.allUsersSQL	SELECT id FROM sysusers
jdbcGroupProvider.userTeacherGroupsSQL	SELECT DISTINCT (subject) FROM teacher_subject WHERE academicyear = ? AND teacher = ?
jdbcGroupProvider.userStudentGroupsSQL	SELECT DISTINCT(subject.id) FROM groups, subject WHERE groups.subject = subject.id AND groups.academicyear = ? AND groups.id IN (SELECT `group` FROM studentgroups WHERE student = ?)
jdbcGroupProvider.loadMembersTeacherSQL	SELECT teacher FROM teacher_subject WHERE academicyear = ? AND subject = ?
jdbcGroupProvider.loadMembersStudentSQL	SELECT student FROM studentgroups WHERE `group` IN (SELECT groups.id FROM groups, subject WHERE groups.academicyear = ? AND groups.subject = ?)
jdbcGroupProvider.loadMembersSQL	SELECT login FROM sysusers WHERE sysusers.id IN (SELECT student FROM studentgroups WHERE `group` IN (SELECT groups.id FROM groups WHERE groups.academicyear = ? AND groups.subject=?))
jdbcGroupProvider.loadAdminsSQL	SELECT student FROM studentgroups WHERE `group` IN (SELECT id FROM groups WHERE name=?)
jdbcGroupProvider.groupCountSQL	SELECT count(DISTINCT(subject)) FROM groups WHERE kind = 0 AND academicyear = ?
jdbcGroupProvider.descriptionSQL	SELECT subject.name FROM subject, groups, programme WHERE groups.subject = subject.id AND subject.programme = programme.id AND groups.academicyear = ? AND subject.id=?
jdbcGroupProvider.allGroupsSQL	SELECT DISTINCT(subject.id) FROM groups, subject, programme WHERE groups.subject = subject.id AND subject.programme = programme.id AND groups.kind = 0 AND groups.academicyear = ?
jdbcGroupProvider.academicYear	2010

Figura 46 - Parámetros del servidor Openfire adaptador a Tutor

4.6.2 Identificación del usuario de Tutor en el servidor Openfire

Otro de los problemas que se ha tenido que resolver es el de cómo realizar la autenticación en el servidor Openfire de forma automática cuando el usuario está autenticado en Tutor. El proceso de esta operación se explica mediante la Figura 47.

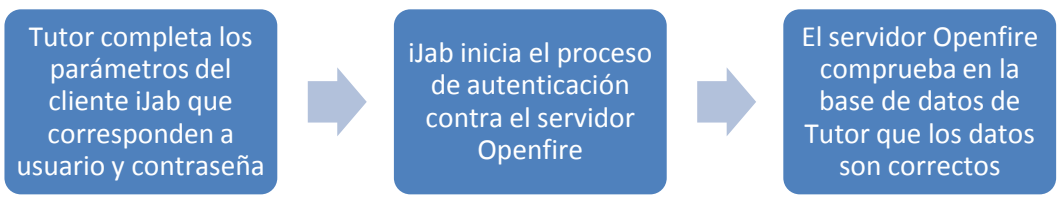


Figura 47 - Proceso de autenticación Tutor-iJab-Openfire

El problema radica en que la autenticación del cliente iJab en el servidor Openfire es un proceso totalmente independiente a la autenticación del usuario en Tutor. De hecho, el cliente podría ser configurado para iniciar sesión en servidores de terceros, tales como los servidores de Facebook o los servidores de Google Talk.

En este proceso intervienen 3 elementos: Tutor, iJab y Openfire.

El primero de ellos, Tutor, mantiene un sistema de autenticación basado en *cookies*. Cuando un usuario se autentica en Tutor se genera en el sistema un *hash de conexión único*, que consiste en una cadena de 64^{40} posibilidades. Este *hash*, junto con el identificador del usuario, se envía como una cookie al navegador del usuario y se almacena con encriptación SHA-1 en la base de datos. El usuario permanecerá autenticado mientras no expire esta *cookie*. Una de las novedades introducidas en Tutor durante el periodo de redacción de esta memoria ha sido la adición de la característica "Recordar mi cuenta", Figura 48. Mediante esta opción, las *cookies* de autenticación se envían con un periodo de expiración de un año, en vez del periodo de dos horas con el que se envían si no se hace uso de ella.

Figura 48 - Opción de "Recordar mi cuenta" en el cuadro de autenticación

El segundo elemento, el cliente iJab, permite realizar la autenticación contra el servidor Openfire haciendo uso de dos cookies, cuyo nombre se les especifica mediante el fichero de configuración *ijab_config.js*. Según la guía escrita por el desarrollador de iJab [61], los parámetros que permiten hacer uso de este sistema autenticación son los siguientes:

- ***auto_login:{true|false}.*** *Habilita la autenticación* automática mediante *cookies*.
- ***username_cookie_field:"username"***. *Si auto_login* está activo, iJab tratará de recuperar el nombre de usuario de la *cookie* con el nombre especificado *por este* parámetro.
- ***token_cookie_field:"SID"***. Si *auto_login* está activo, iJab tratará de recuperar la clave de autenticación de la *cookie* con el nombre especificado por este parámetro.

Para el caso de Tutor, las *cookies* que almacenan el identificador de usuario y el *hash* de conexión son *TUTOR_id* y *TUTOR_hash*.

Por último, el servidor Openfire debe conocer cómo realizar la autenticación una vez recibe el nombre de usuario y la clave de autenticación especificados por Tutor. Para ello, permite configurar dos parámetros que ya aparecieron en la sección 4.6.1, *jdbcAuthProvider.passwordSQL* y *jdbcAuthProvider.passwordType*.

El proceso que realiza Openfire consiste en recuperar de la base de datos la contraseña del usuario que intenta autenticarse, haciendo uso de la consulta SQL especificada en el parámetro *jdbcAuthProvider.passwordSQL*. De existir una contraseña para el usuario, una vez recuperada esta contraseña de la base de datos, la compara con la contraseña con la que se solicitó la petición de autenticación.

En el caso de Tutor, lo que se compara no es realmente una contraseña, pues se ha considerado inseguro enviar la contraseña, aún estando encriptada, al navegador. En su lugar, es el *hash* de conexión comentado anteriormente lo que se compara. Dicho *hash* se almacena en la base de datos encriptado mediante SHA-1.

El proceso completo puede verse en la Figura 49, que muestra cómo queda resuelto el problema de autenticación en el servidor Openfire para los usuarios de Tutor.

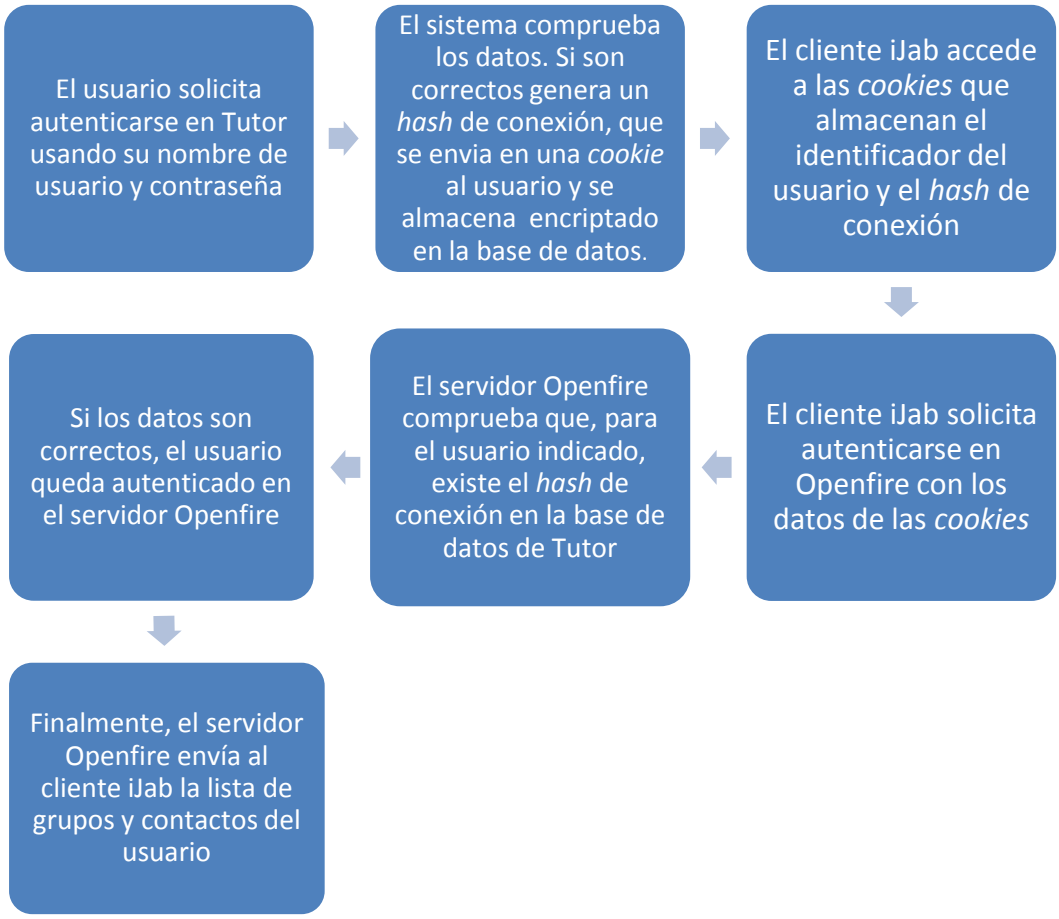


Figura 49 - Proceso completo de autenticación en Openfire

4.6.3 Problemas por resolver

Existen dos problemas por resolver en cuanto a la integración de la funcionalidad de chat en Tutor, el primero de ellos es un problema de lentitud a la hora de cargar los datos por parte del cliente, mientras que el segundo es un problema de restricción de uso.

4.6.3.1 Problema de lentitud de carga en el cliente

Para entender este problema, hay que detenerse a observar la Figura 50. En esta figura se puede ver la interfaz del chat, una vez integrado en Tutor, para un usuario de tipo profesor. En este ejemplo, el número de contactos asciende a la cantidad de 265, pero podrían ser más. Esto implica que se está recuperando la información de 265 contactos y generando de forma dinámica el contenido HTML necesario para mostrar cada uno de los elementos de interfaz que corresponden a un usuario. En las pruebas realizadas, este proceso no requiere más de 4 segundos. El problema es que este proceso se debe realizar cada vez que el usuario cambia de página, lo cual afecta negativamente a la navegación en el sistema.

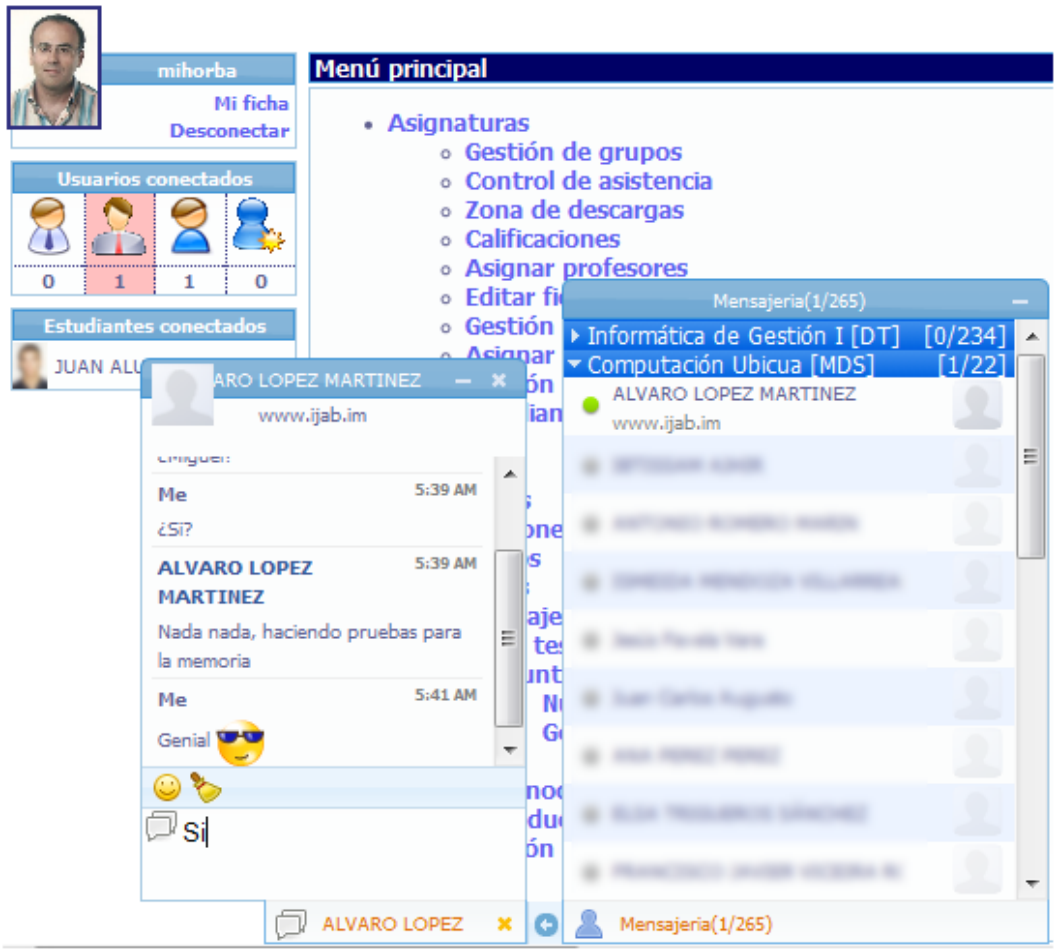


Figura 50 - Interfaz del chat para un profesor

Existen dos soluciones posibles que resolverían, o al menos paliarían, los efectos de este problema. La primera de ellas pasa por usar patrones de diseño AJAX tales como *IFrame Call* [19] en la totalidad de Tutor. La otra solución pasa por modificar íntegramente el funcionamiento del cliente iJab. En cualquier caso, ambas soluciones no son excluyentes.

El usar patrones de diseño AJAX a nivel de sistema en Tutor acercaría la plataforma a otras plataformas como Facebook o Tuenti. El uso del patrón *IFrame Call* en estas plataformas permite que la comunicación mediante el navegador y el servidor se haga mediante un elemento IFrame [20] oculto, obteniendo la información solicitada y *sustituyendo sólo la parte de la página que debe actualizarse*, sin efectuar un cambio de página completo en el navegador. Este patrón ya se usa en algunas de las funcionalidades de Tutor, por ejemplo: en la *Zona de Descargas*, a la hora de realizar la subida de un fichero, o en el *Gestor de Calificaciones*, cuando se obtiene la lista de estudiantes de una asignatura. La diferencia con Facebook o Tuenti reside en que estas plataformas usan este tipo de patrones a nivel de sistema. Esto implica que el cambiar de una funcionalidad a otra, no requiere un cambio de página. De esta manera, la barra de chat permanece constante en la parte inferior de la página, y sólo requiere ser cargada inicialmente. Este tipo de solución genera una serie de problemas adicionales. Uno de los más importantes es que, dado que que en cierto modo se está "engañando" al navegador y no se están produciendo cambios de página, éste no puede reflejar la navegación en el historial. Para resolver estos problemas adicionales existen otro tipo de patrones, como el patrón *Unique URLs* [51]. Mediante este patrón se le da una dirección única a cada una de las funcionalidades u operaciones que se realizan sobre el sistema, aunque no se haya producido un cambio de página.

Un ejemplo de uso de este patrón se encuentra en la funcionalidad de *Mensajería interna* explicada en el Capítulo 3. En esta funcionalidad, se le asigna una dirección única a cada una de las cadenas de mensajes, aun a pesar de que no se realice un cambio de página en el navegador (Figura 51). De este modo, se puede usar la dirección única para acceder de forma directa a una cadena de mensajes.

En principio, aplicar esta solución implicaría realizar cambios en gran parte de Tutor, afectando a todas las funcionalidades existentes. Por otra parte, se daría el caso de un uso recursivo del patrón IFrame Call, existiendo funcionalidades usando IFrames dentro de un IFrame superior. Este uso recursivo provocaría que estas funcionalidades dejasen de actuar correctamente. Es por ello que no se ha aplicado en la resolución del problema, si bien la idea puede ser recuperada y desarrollada en el futuro.

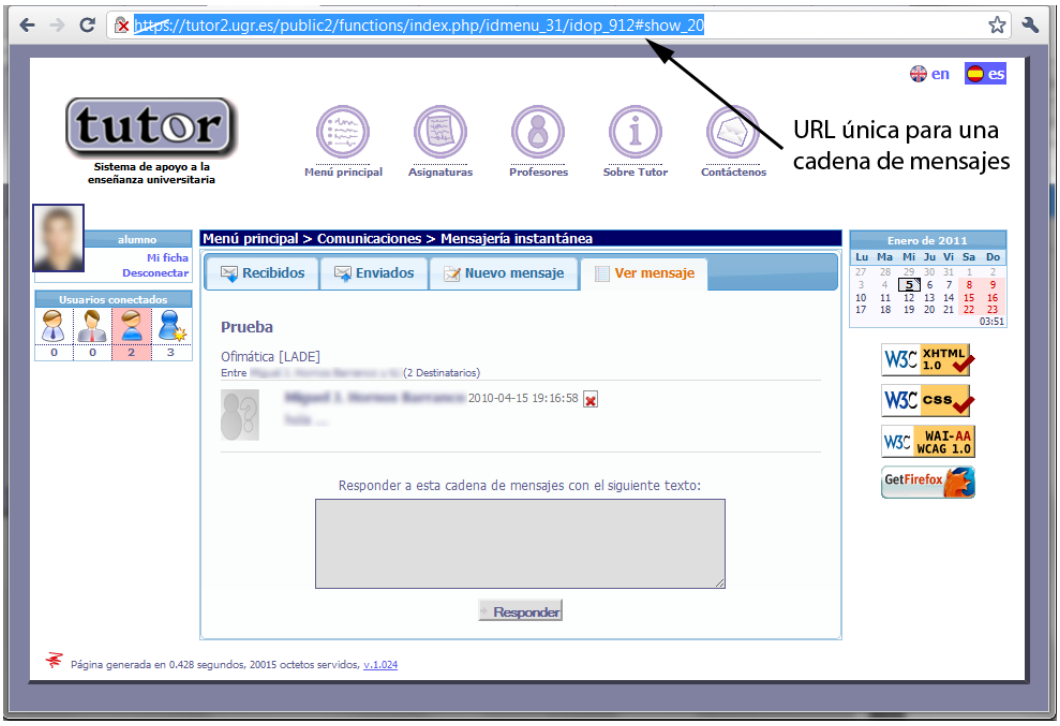


Figura 51 - Utilización del patrón Unique URLs en la funcionalidad de Mensajería Interna de Tutor

La otra posibilidad pasa por modificar el cliente iJab para recuperar la información y generar el contenido HTML sólo para los usuarios conectados en ese momento. Esta solución que, en un primer momento, puede parecer trivial, dada la disponibilidad del código fuente del cliente, no lo es tanto, una vez se comprende el funcionamiento interno del cliente iJab y del protocolo XMPP.

Tanto el cliente como el protocolo parten de la base de que el cliente recupera, en el proceso de conexión, la información correspondiente a todos sus contactos, incluyendo el estado de "presencia" de éstos. A partir de ese momento inicial, el servidor sólo se encarga de notificar al cliente los cambios de estado que se producen en los contactos, como, por ejemplo, los cambios de conectado a desconectado y viceversa.

Por tanto, el cambio propuesto implica modificaciones a nivel de protocolo, lo cual complica la viabilidad del mismo.

Una solución intermedia pasa por recuperar la información de los contactos de un usuario, pero generando sólo los elementos HTML correspondientes a los usuarios conectados. Sin embargo, el modo de funcionamiento del cliente iJab es similar al del protocolo XMPP: inicialmente se generan todos los elementos HTML y, a partir de ese momento, se actualizan los elementos HTML con los cambios de estado producidos. Esto complica realizar la modificación propuesta, pues habría que rediseñar gran parte del funcionamiento del cliente.

En el momento de la escritura de esta memoria, los responsables del proyecto iJab han dejado temporalmente de lado el cliente de código abierto, centrándose en el desarrollo de un producto de cliente online. No obstante, el problema descrito en esta memoria ha sido notificado por varios usuarios, lo cual permite pensar que en un futuro será tratado.

4.6.3.2 Problema de restricción de uso

Existe otro problema asociado al uso del Chat cuyas características responden más al posible uso que se le puede dar en la plataforma que a temas de diseño o implementación.

Dado que existen funcionalidades de Tutor que permiten realizar tareas tales como realizar exámenes, se hace necesario definir una forma de deshabilitar el acceso al uso del chat durante las tareas que requieran ser realizadas por el alumno de forma individual. Ante este problema, se pueden dar varias soluciones, no excluyentes entre sí. Una solución sería que el profesor de un grupo de alumnos pudiera restringir el uso del chat mientras estime que es necesario adoptar esta medida. Esta medida implicaría estudiar casos como el de un alumno compartido por varios profesores, pues no tendría sentido que la restricción aplicada por uno pudiera ser eliminada por otro.

Otra posible solución sería que determinadas funcionalidades deshabilitarán el uso del chat mientras se hace uso de ellas. A este respecto, también podría darse la particularidad de funcionalidades que deshabiliten la posibilidad de chatear entre estudiantes, pero que sí permitiesen una comunicación con el profesor.

4.7. Futuras mejoras

El hecho de usar un protocolo existente y tan potente como es XMPP permite pensar en muchas posibilidades de cara al futuro. Adicionalmente, usar un servidor libre y en desarrollo como Openfire abre nuevas puertas.

Por ejemplo, es fácil pensar en usar un cliente diferente a iJab que permita mayores posibilidades, tales como conversaciones vía voz, tal y como permite Google Chat. De hecho, es posible autenticarse en el servidor Openfire instalado en Tutor mediante un cliente XMPP/Jabber tradicional y hacer uso de las ventajas que éste pueda aportar respecto a un cliente web. Quizás el desarrollo de un cliente web XMPP con soporte a audio/video podría ser un buen proyecto fin de carrera para futuros estudiantes, pues tras una larga búsqueda no he encontrado ninguno que cumpla con esta característica.

También sería posible aprovechar algunos de los plugins disponibles para Openfire, como por ejemplo el plugin de soporte SIP [45] para realizar llamadas desde Tutor a los teléfonos universitarios fijos que trabajan sobre tecnología IP.

Por otra parte, aunque en un principio se pretendía desarrollar la funcionalidad de chat para permitir comunicaciones entre dos usuarios, aprovechando, una vez más, las posibilidades del protocolo XMPP y del servidor Openfire, se podrían realizar conversaciones multiusuario de forma sencilla.

Para concluir, durante el desarrollo de este proyecto se amplió el elemento de *Conciencia de grupo* que aparece en cada una de las páginas del sistema (Figura 52) para mostrar los profesores y estudiantes que comparten alguna asignatura con el usuario y están online en ese momento. Vincular este elemento con el cliente iJab para poder iniciar conversaciones es otra de las posibles futuras ampliaciones.

Figura 52 - Elemento de conciencia de grupo en Tutor

Capítulo 5. Estadísticas y uso del sistema

5.1. Motivación

Con el fin de estudiar la evolución del uso de las diferentes funcionalidades de Tutor a lo largo de los años, se ha desarrollado un módulo de estadísticas. Si bien este tipo de análisis se puede llevar a cabo a través de consultas realizadas en la propia base de datos, el desarrollo de un módulo que implemente dichas consultas y las muestre de forma ordenada supone una serie de ventajas frente a realizar el análisis manualmente. Adicionalmente, hacer este módulo accesible para el administrador desde la propia interfaz del sistema facilita la tarea de consultar los datos analizados.

Por último, dada la nueva arquitectura modular del sistema, desarrollar un módulo de estadísticas permite probar y demostrar una de las ventajas de esta arquitectura para el desarrollo de funcionalidades que interactúan con otros módulos del sistema.

5.2. Especificación de requisitos

5.2.1 Requisitos funcionales

Dadas las características del módulo a desarrollar, existe un solo requisito funcional que debe ser cumplido:

Consulta de estadísticas

El módulo a desarrollar debe mostrar de forma ordenada y clara una serie de estadísticas asociadas al sistema. Esta serie de estadísticas estarán definidas por los diferentes módulos del sistema que estén en uso en ese momento. Los datos deben poder ordenarse y ser consultados cronológicamente, en el caso de que los datos permitan esta posibilidad.

5.2.2 Requisitos no funcionales

El principal requisito no funcional de este módulo es la extensibilidad.

Extensibilidad

El módulo a desarrollar debe evolucionar en función de los módulos que se añadan o eliminen del sistema. También debe evolucionar en función del transcurso de los años, sin que, por ejemplo, un cambio de curso académico, suponga la reconfiguración del módulo. De este modo, según transcurran los años en Tutor, irán apareciendo automáticamente los datos asociados a estos nuevos años, sin tener que cambiar ningún fragmento del código.

5.2.3 Casos de uso

Sólo existe un caso de uso para esta funcionalidad, el caso de uso *Consultar Estadísticas*.

Caso de Uso	Consultar estadísticas				CU-1
Actores	Administrador				
Tipo	Primario, Esencial				
Dependencia					
Precondición	El usuario está autenticado en Tutor y autorizado para realizar esta operación.				
Postcondición					
Autor	Álvaro López Martínez	**Fecha**	1-09-2010	**Versión**	1.0

Propósito
Consultar las estadísticas del sistema

Resumen
El usuario consulta las estadísticas del sistema

Curso Normal			
1	En el menú principal, el usuario acciona la opción *Estadísticas*.	**2**	El sistema muestra un listado de todas las estadísticas presentes en el sistema.

Cursos Alternos	

Comentarios

5.3. Análisis y diseño

Crear un módulo de estadísticas era, a priori, una tarea sencilla. Realmente generar las estadísticas era cuestión de diseñar las consultas SQL necesarias en la base de datos para obtener los datos y presentar éstos mediante un diseño organizado.

Ahora bien, para realizar este módulo se presentaban varias aproximaciones. La primera de ellas consistía en integrar las consultas SQL en el propio módulo de estadísticas. De esta forma, el módulo realizaría las consultas, y la presentación de los datos se haría a medida de cada una de las estadísticas analizadas.

Sin embargo, esta primera aproximación no cumple el requisito no funcional de la extensibilidad, pues si en un futuro se añadieran nuevas funcionalidades a Tutor, se haría necesario modificar el módulo para obtener las estadísticas que pudieran estar asociadas a este módulo.

Por ello, se diseñó una segunda aproximación, en la que, reaprovechando el trabajo desarrollado en el Capítulo 1, se le daba la responsabilidad a cada uno de los módulos de Tutor de presentar las estadísticas asociadas al mismo.

El proceso asociado a esta aproximación se puede ver en la Figura 53.

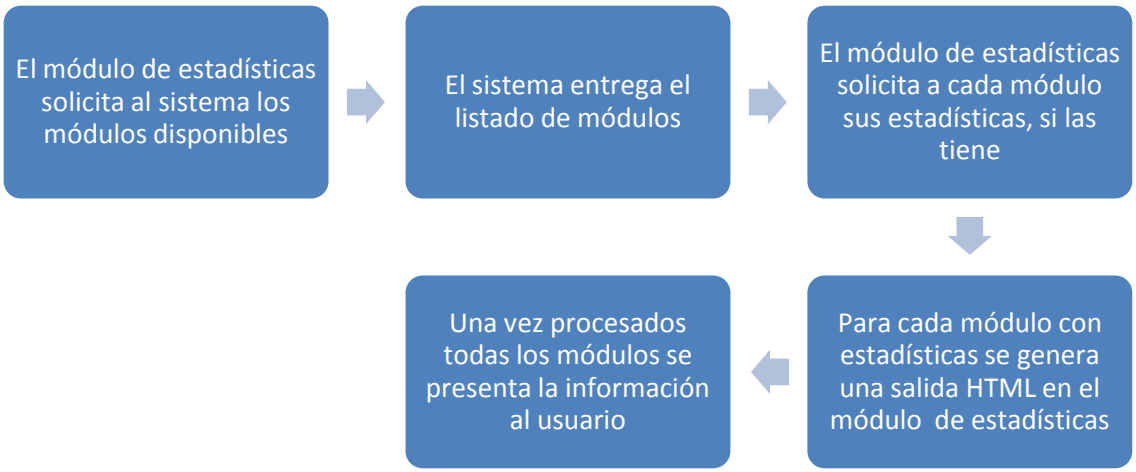


Figura 53 - Proceso de generación de estadísticas

Esta aproximación, que cumplía con el requisito de extensibilidad, presentaba otro problema, el de la estructura de los datos analizables. Por ejemplo, del módulo de *Gestión de Grupos* podía ser interesante obtener los datos de número de estudiantes matriculados por asignatura y curso, y los totales por curso. Sin embargo, en el caso del módulo de *Preferencias de usuario* lo interesante podía ser el número de usuarios que eligen entre un tema u otro, independientemente de asignatura, curso académico o cualquier otra variable.

En la Figura 54 y en la Figura 55 se puede ver gráficamente el problema del que se está hablando, diferentes datos estadísticos implican diferentes variables. Una posible solución a este problema era que cada módulo generase la salida HTML correspondiente a las estadísticas que ofrecía. Sin embargo, esta solución implicaría que si, en un momento dado, se quisiera cambiar la presentación de los datos, el control ya no residiría sobre el módulo de estadísticas, sino sobre cada uno de los módulos del sistema.

Por tanto, el problema que se planteaba era cómo diseñar un sistema capaz de generar tablas estadísticas cuyos encabezados, pies de tabla o últimas columnas podían diferir totalmente de unos datos estadísticos a otros, sin que la responsabilidad recayese sobre cada uno de los módulos del sistema.

Estudiantes por asignatura y curso académico

Concepto	2007	2008	2009	2010	Total
Informática Aplicada a la Gestión de la Empresa [DCE]		389	413		802
Ofimática [LADE]	225	233	216		674
Informática de Gestión I [DT]		191	204	223	618
Fundamentos del Software [GII]				505	505
Sistemas Operativos I [ITIS]		175	137	172	484
Sistemas Operativos II [ITIG]		178	156		334
Sistemas Operativos I [II]		89	101	128	318
Sistemas Operativos I [ITIG]			140	157	297
Sistemas Operativos II [ITIS]		153	137		290
Programación Dirigida a Objetos [II]		121	66	82	269
Programación Concurrente [II]		123	115		238
Ofimática Empresarial [DCE]	72	87	71		230
Informática Aplicada a la Gestión Comercial [LITM]	75	67	85		227
Ingeniería del Software I [II]		68	51	71	190
Descripción de Lenguajes de Programación [II]		58	52	73	183
Sistemas Operativos II [II]		83	95		178
Diseño de Sistemas Operativos [II]		73	69		142
Informática de Gestión II [DT]		62	45		107
Sistemas Informáticos Distribuidos [ITIS]		38	28	7	73
Procesos de Creación Artística, Audiovisual y Multimedia [LBA]		34	34		68
Informática Gráfica [II]			65		65
Ampliación de Ingeniería del Software [ITIS]			61		61
Curso de Prueba [LADE]	60				60
Asignatura de Prueba [LADED]	21	17	8	6	52
Análisis y Usos Geográficos de la Información [LG]		13	31		44
Computación Ubicua [MDS]		12	13	18	43
Metodología de Investigación [MDS]			14	27	41
Demo Proyecto [II]	31				31
Sistemas Hipermedia [MDS]			15	16	31
Digitalización 3D [MDS]			8	10	18
Ingeniería Web [MDS]			17		17
Modelado y Visualización de Volúmenes [MDS]			7	10	17
Metodologías y Herramientas para el Desarrollo Evolutivo de Software [MDS]			16		16
Gestión de Redes [IT]		15			15
Técnicas Avanzadas de Modelado de Sólidos [MDS]			6	9	15
Fundamentos de Geometría y Geometría Computacional [MDS]			3	11	14
Ingeniería de la Usabilidad y Ética Informática [MDS]			13		13
Sistemas Colaborativos y Procesos de Negocios [MDS]			13		13
Web semántica [MDS]			11		11
Realidad Virtual [MDS]			10		10
Almacenes de Datos y Sistemas OLAP [MDS]			8		8
Visualización Expresiva y Animación [MDS]			6		6
Total	484	2279	2540	1525	6828

Figura 54 - Tabla de estadísticas para estudiantes por asignatura y curso académico

Preferencias				
Concepto	Administradores	Profesores	Estudiantes	Total
redmond		48	2698	2746
start			9	9
black-tie			9	9
sunny			9	9
dark-hive			8	8
south-street	1		8	9
cupertino			8	8
blitzer			6	6
le-frog			5	5
excite-bike			5	5
trontastic			4	4
dot-luv			3	3
vader			3	3
ui-darkness			3	3
hot-sneaks			2	2
ui-lightness			2	2
flick			1	1
eggplant			1	1
overcast			1	1
swanky-purse			1	1
Total	1	48	2786	2835

Figura 55 - Tabla de estadísticas para el uso de temas gráficos por tipo de usuario

La solución finalmente adoptada pasa por definir una serie de parámetros comunes a todas las estadísticas. Son los siguientes:

- **Título del concepto**: Título de los datos presentados en la tabla. En el caso de la Figura 55 sería "Preferencias".
- **Claves**: Vector con cada uno de los títulos para las columnas que se presentan en la tabla. En el caso de la Figura 55, este vector contendría los siguientes datos: Administradores, Profesores, Estudiantes y Total.
- **Datos:** Matriz bidimensional que contiene cada uno de los conceptos presentados en la tabla y los valores asociados a cada una de las columnas. En el caso de la Figura 55, la representación sería la siguiente:

$$Datos = \begin{Bmatrix} \{redmond \quad \{,48,2698,2746\}\} \\ \{start \quad \{,,9,9\}\} \\ \{\dots \quad \{\dots,\dots,\dots,\dots\}\} \\ \{swanky-purse \quad \{,,1,1\}\} \end{Bmatrix}$$

- **Última fila**: Vector con cada uno de los valores para el pie de tabla. Este valor es opcional, pues no tiene que mostrarse necesariamente un pie de tabla, como ocurre en el caso de las estadísticas relativas al número de usuarios conectados recientemente, representadas en la Figura 56. Sin embargo, en la Figura 55 sí se muestra esta última fila con los valores totales para cada columna, donde estos valores son: 1, 48, 2786 y 2835.

Últimos usuarios				
Concepto	Administradores	Profesores	Estudiantes	Total
Usuarios en la última hora	1	1	4	6
Usuarios en las últimas 24 horas	1	3	137	141
Usuarios en los últimos 7 días	1	10	522	533

Figura 56 - Tabla de estadísticas para el número de usuarios conectados recientemente

Definidos estos parámetros, la responsabilidad de la generación de las visualizaciones recae sobre el módulo de estadísticas. Para ello, se aprovecha el potencial de la clase *Template* que, haciendo uso de una plantilla genérica, y mediante los datos ofrecidos por cada módulo, construye las tablas a medida.

5.3.1 Diagrama de clases

Todas las operaciones que realiza este módulo se encuentran en la clase *StatisticsManager*. Esta clase se relaciona con la clase *Tutor,* que a su vez se relaciona con todos los módulos del sistema. Como se ha explicado en la sección 5.3, la clase *StatisticsManager*, mediante la operación *getHTML*, solicita a *Tutor* los módulos disponibles en el sistema. A los módulos que implementan la operación *getStats* les solicita las estadísticas de éstos, y mediante las funciones *buildModule* y *buildConcept* construye cada una de las tablas que serán mostradas en la plantilla *sheet*. La Figura 57 corresponde al diagrama que modela este comportamiento.

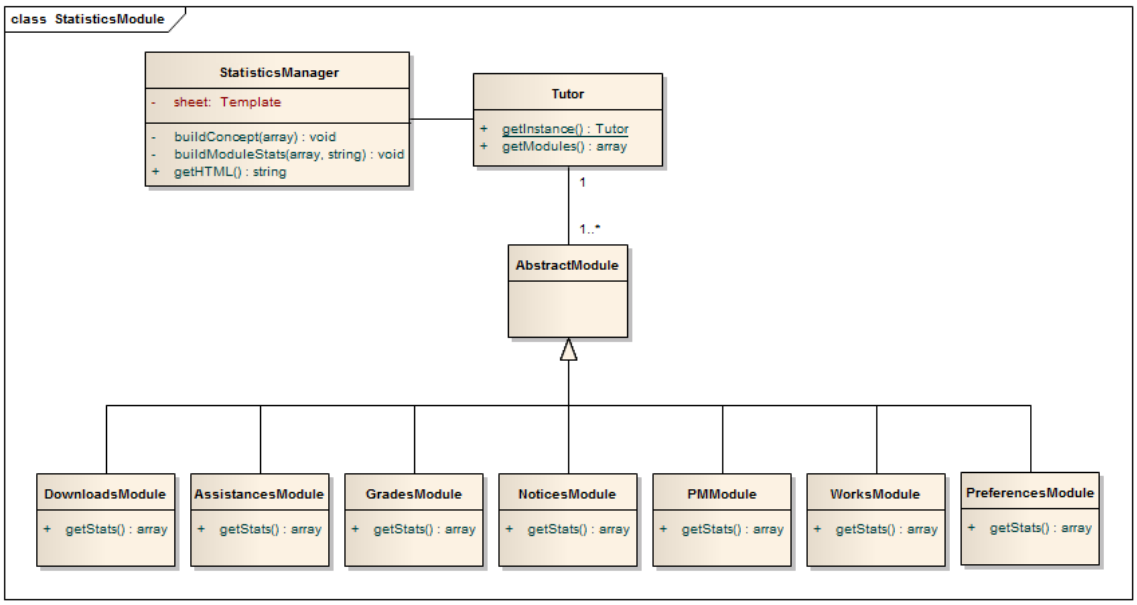


Figura 57 - Diagrama de clases del módulo de Estadísticas

Si bien hay actualmente más módulos en el sistema, sólo se muestran en el diagrama aquellos que implementan la función *getStats*, que son los que se muestran en la fila inferior de la Figura 57.

5.4. Interfaz de usuario

Esta funcionalidad, que permite el acceso a una serie de datos estadísticos sobre la utilización del sistema, está reservada a aquellos usuarios que desempeñen el rol de administrador. La interfaz del módulo que implementa dicha funcionalidad se muestra en la Figura 58. Como se puede apreciar en ella, al término de esta memoria se ofrecen datos estadísticos que se recogen en 17 tablas, divididas en 8 secciones o apartados, generalmente referidas a una funcionalidad diferente. Las tablas relativas a "Estudiantes por asignatura y curso académico", "Preferencias" y "Últimos usuarios" ya se han mostrado anteriormente, en las Figura 54, Figura 55 y Figura 56 respectivamente.

Figura 58 - Interfaz del módulo de Estadísticas

A continuación se muestran cada una de las figuras correspondientes a las 14 tablas restantes, comentándose brevemente cada una de ellas.

La Figura 59 muestra el top 30 de usuarios por número de accesos desde que se empezó a contabilizar el mismo, en noviembre del 2010. La tabla no hace uso de datos de pie de tabla y sólo tiene una columna de datos. De modo similar, la Figura 60 muestra el top 30 de usuarios por número de trabajos entregados.

Top 30 de usuarios por número de accesos

Nombre del usuario	Accesos
[redacted]	441
[redacted]	328
[redacted]	191
[redacted]	172
[redacted]	170
[redacted]	161
[redacted]	157
[redacted]	145
[redacted]	141
[redacted]	139
[redacted]	137
[redacted]	104
[redacted]	104
[redacted]	104
[redacted]	102
[redacted]	102
[redacted]	102
[redacted]	100
[redacted]	99
[redacted]	94
[redacted]	93
[redacted]	92
[redacted]	92
[redacted]	91
[redacted]	88
[redacted]	88
[redacted]	86
[redacted]	84
[redacted]	81
[redacted]	80

Figura 59 - Estadísticas: Top 30 de usuarios por número de accesos

Top 30 trabajos entregados por estudiante

Concepto	Total
[redacted]	31
[redacted]	30
[redacted]	29
[redacted]	29
[redacted]	27
[redacted]	27
[redacted]	26
[redacted]	26
[redacted]	25
[redacted]	25
[redacted]	25
[redacted]	23
[redacted]	23
[redacted]	23
[redacted]	23
[redacted]	23
[redacted]	22
[redacted]	22
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21
[redacted]	21

Figura 60 - Estadísticas: Top 30 de número de trabajos entregados por estudiantes

La Figura 61 y la Figura 62 muestran las estadísticas correspondientes a trabajos propuestos por profesor y por asignatura en cada curso académico. Estas tablas sí hacen uso del pie de tabla, para mostrar el total de trabajos por curso.

Trabajos por profesor y curso académico

Concepto	2007	2008	2009	2010	Total
[redacted]				36	36
[redacted]		22	26	7	55
[redacted]		12	22	7	41
[redacted]		20	18	12	50
[redacted]		18	5		23
[redacted]		17	12	12	41
[redacted]	1	11	16	2	30
[redacted]		14		4	18
[redacted]		9	14	7	30
[redacted]		10	13	8	31
[redacted]		1	13	4	18
[redacted]				12	12
[redacted]		11	7	4	22
[redacted]		4	7	11	22
[redacted]	1	10	9	7	27
[redacted]		9	4	2	15
[redacted]				6	6
[redacted]			6		6
[redacted]				6	6
[redacted]		6	5		11
[redacted]			4	3	7
[redacted]				4	4
[redacted]		4			4
[redacted]				3	3
[redacted]	2	3			5
[redacted]	2	2	3	2	9
[redacted]		3			3
[redacted]		2	2		4
[redacted]	2	2	1		5
[redacted]		1	2		3
[redacted]				1	1
[redacted]		1			1
Total	8	192	189	160	549

Figura 61 – Estadísticas: Trabajos propuestos por profesor y curso académico

Trabajos por asignatura y curso académico

Concepto	2007	2008	2009	2010	Total
Fundamentos del Software [GII]				107	107
Informática Aplicada a la Gestión de la Empresa [DCE]		28	21	1	50
Programación Dirigida a Objetos [II]		25	24	10	59
Informática de Gestión I [DT]		15	18	23	56
Sistemas Operativos II [ITIS]		20	11		31
Ingeniería del Software I [II]		19	14	2	35
Programación Concurrente [II]		17	12		29
Ofimática Empresarial [DCE]	4	16	10		30
Informática Aplicada a la Gestión Comercial [LITM]	1	11	15		27
Ampliación de Ingeniería del Software [ITIS]			12		12
Procesos de Creación Artística, Audiovisual y Multimedia [LBA]		7	9		16
Sistemas Operativos I [ITIG]			8	1	9
Sistemas Operativos I [ITIS]		6	8	4	18
Sistemas Operativos II [ITIG]		8			8
Sistemas Operativos II [II]		6	7		13
Sistemas Operativos I [II]		4	4	3	11
Descripción de Lenguajes de Programación [II]		3	4	2	9
Asignatura de Prueba [LADED]		4	4	2	10
Informática Gráfica [II]			3		3
Ofimática [LADE]	2	1	2		5
Fundamentos de Geometría y Geometría Computacional [MDS]				1	1
Sistemas Informáticos Distribuidos [ITIS]		1	1	1	3
Realidad Virtual [MDS]			1		1
Sistemas Hipermedia [MDS]				1	1
Diseño de Sistemas Operativos [II]				1	1
Modelado y Visualización de Volúmenes [MDS]			1	1	2
Demo Proyecto [II]	1				1
Gestión de Redes [IT]		1			1
Total	8	192	189	160	549

Figura 62 – Estadísticas: Trabajos propuestos por asignatura y curso académico

La Figura 63 muestra la estadística de mensajes enviados por asignatura mediante el módulo de *Mensajería interna*. En esta tabla sólo se contemplan los datos correspondientes al curso académico 2010/2011, pues es el primer curso en el que la funcionalidad ha estado disponible. El cumplir el requisito de extensibilidad descrito en el apartado 5.2.2 permite que esta tabla se construya automáticamente, pese a que los datos difieran con el resto de estadísticas en el rango de cursos académicos con información disponible.

Mensajes por asignatura y curso académico

Concepto	2010	Total
Fundamentos del Software [GII]	265	265
Ofimática [LADE]	240	240
Informática Aplicada a la Gestión de la Empresa [DCE]	95	95
Informática Aplicada a la Gestión Comercial [LITM]	86	86
Informática de Gestión I [DT]	51	51
Programación Concurrente [II]	31	31
Digitalización 3D [MDS]	24	24
Sistemas Operativos I [ITIS]	21	21
Ofimática Empresarial [DCE]	18	18
Sistemas Hipermedia [MDS]	12	12
Sistemas Operativos I [II]	11	11
Metodología de Investigación [MDS]	10	10
Sistemas Operativos II [ITIS]	10	10
Ingeniería del Software I [II]	10	10
Sistemas Operativos I [ITIG]	10	10
Asignatura de Prueba [LADED]	9	9
Informática Gráfica [II]	6	6
Ampliación de Ingeniería del Software [ITIS]	4	4
Sistemas Operativos II [II]	3	3
Programación Dirigida a Objetos [II]	3	3
Computación Ubicua [MDS]	2	2
Fundamentos de Geometría y Geometría Computacional [MDS]	2	2
Sistemas Operativos II [ITIG]	1	1
Total	924	924

Figura 63 - Estadísticas: Mensajes por asignatura y curso académico

La Figura 64 y la Figura 65 muestran respectivamente los datos estadísticos correspondientes al número de avisos por profesor y por asignatura según el año en el que fueron publicados. Ambas estadísticas pertenecen al módulo de *Avisos*.

Avisos por profesor y año

Concepto	2008	2009	2010	Total
[redacted]	8	42	43	93
[redacted]	2	32	41	75
[redacted]	4	21	32	57
[redacted]	4	24	25	53
[redacted]	6	16	14	36
[redacted]	2	3	28	34
[redacted]	5	12	14	31
[redacted]		12	14	26
[redacted]	1	13	12	26
[redacted]	2	11	13	26
[redacted]		14	8	22
[redacted]	5	9	6	20
[redacted]	3	4	11	18
[redacted]		2	13	15
[redacted]	4	7	3	14
[redacted]			12	12
[redacted]		2	7	9
[redacted]		3	6	9
[redacted]	1	3	5	9
[redacted]			8	8
[redacted]			8	8
[redacted]		2	5	8
[redacted]		7		7
[redacted]		1	6	7
[redacted]		5	1	6
[redacted]	1	4	1	6
[redacted]	4	2		6
[redacted]			4	4
[redacted]			4	4
[redacted]			4	4
[redacted]			3	3
[redacted]			3	3
[redacted]			3	3
[redacted]		2		2
[redacted]			2	2
[redacted]			2	2
[redacted]		1		1
Total	52	254	361	669

Figura 64 - Estadísticas: Avisos por profesor y año

Avisos por asignatura y año				
Concepto	2008	2009	2010	Total
Informática Aplicada a la Gestión de la Empresa [DCE]		22	40	62
Sistemas Operativos I [ITIS]	8	21	28	58
Informática de Gestión I [DT]	9	21	17	47
Sistemas Operativos II [ITIS]		21	23	44
Fundamentos del Software [GII]			34	34
Programación Dirigida a Objetos [II]	2	19	12	33
Informática Aplicada a la Gestión Comercial [LITM]	5	16	10	31
Sistemas Operativos II [II]		16	14	30
Ofimática Empresarial [DCE]	4	12	12	28
Diseño de Sistemas Operativos [II]		12	15	27
Sistemas Operativos I [ITIG]		5	21	26
Sistemas Operativos I [II]	3	10	12	25
Ofimática [LADE]	4	10	10	24
Programación Concurrente [II]		11	9	20
Ingeniería del Software I [II]	6	9	5	20
Informática de Gestión II [DT]		9	6	15
Procesos de Creación Artística, Audiovisual y Multimedia [LBA]	4	7	4	15
Sistemas Informáticos Distribuidos [ITIS]	2	5	6	14
Informática Gráfica [II]			12	12
Sistemas Operativos II [ITIG]		6	6	12
Ampliación de Ingeniería del Software [ITIS]			10	10
Sistemas Hipermedia [MDS]		2	7	9
Técnicas Avanzadas de Modelado de Sólidos [MDS]		2	7	9
Gestión de Redes [IT]		7		7
Análisis y Usos Geográficos de la Información [LG]		3	4	7
Modelado y Visualización de Volúmenes [MDS]			6	6
Asignatura de Prueba [LADED]	4		2	6
Sistemas Colaborativos y Procesos de Negocios [MDS]			5	5
Metodologías y Herramientas para el Desarrollo Evolutivo de Software [MDS]			4	4
Fundamentos de Geometría y Geometría Computacional [MDS]			3	3
Digitalización 3D [MDS]			3	3
Descripción de Lenguajes de Programación [II]		1	2	3
Computación Ubicua [MDS]			2	2
Metodología de Investigación [MDS]			1	1
Realidad Virtual [MDS]			1	1
Total	51	247	353	653

Figura 65 - Estadísticas: Avisos por asignatura y año

La Figura 66 y la Figura 67 muestran el número de convocatorias creadas en el módulo de *Calificaciones* para cada profesor y cada asignatura respectivamente, en función del curso académico.

Calificaciones por profesor y curso

Concepto	2007	2008	2009	2010	Total
[blurred]		10	13	10	33
[blurred]			22	5	27
[blurred]	5	9	12	1	27
[blurred]		12	8	3	23
[blurred]		13	9		22
[blurred]		8	11		19
[blurred]		6	4	6	16
[blurred]		9	5	1	15
[blurred]	1	3	3	5	12
[blurred]		6	4		10
[blurred]			8	1	9
[blurred]		5	4		9
[blurred]			6		6
[blurred]		4	2		6
[blurred]		3	1		4
[blurred]				3	3
[blurred]			3		3
[blurred]		2			2
[blurred]			2		2
[blurred]		1	1		2
[blurred]			1	1	2
[blurred]		1			1
[blurred]			1		1
[blurred]		1			1
[blurred]			1		1
[blurred]			1		1
Total	6	93	122	36	257

Figura 66 - Estadísticas: Calificaciones por profesor y curso académico

Calificaciones por asignatura y curso

Concepto	2007	2008	2009	2010	Total
Ingeniería del Software I [II]		15	13	10	38
Sistemas Operativos II [ITIS]		11	13		24
Informática de Gestión I [DT]		12	9	2	23
Informática Aplicada a la Gestión de la Empresa [DCE]		8	14		22
Sistemas Operativos I [ITIS]			14	5	19
Informática Aplicada a la Gestión Comercial [LITM]		10	5		15
Asignatura de Prueba [LADED]		4	5	5	14
Ofimática [LADE]	4	5	5		14
Análisis y Usos Geográficos de la Información [LG]		5	5		10
Sistemas Operativos I [ITIG]			5	5	10
Procesos de Creación Artística, Audiovisual y Multimedia [LBA]		5	3		8
Informática de Gestión II [DT]		3	4		7
Sistemas Informáticos Distribuidos [ITIS]		3	4		7
Descripción de Lenguajes de Programación [II]		4	1	2	7
Sistemas Operativos II [II]			6		6
Fundamentos del Software [GII]				5	5
Programación Concurrente [II]		4	1		5
Sistemas Operativos I [II]		2	2	1	5
Informática Gráfica [II]			4		4
Demo Proyecto [II]	2	1			3
Gestión de Redes [IT]		2			2
Ampliación de Ingeniería del Software [ITIS]			1		1
Visualización Expresiva y Animación [MDS]			1		1
Ofimática Empresarial [DCE]			1		1
Total	6	94	116	35	251

Figura 67 - Estadísticas: Calificaciones por asignatura y curso académico

La Figura 68 muestra el número de ficheros subidos por cada profesor y curso académico, haciendo uso del módulo de *Zona de Descargas*.

Ficheros por profesor y curso académico

Concepto	2007	2008	2009	2010	Total
[redacted]	7	31	26	60	124
[redacted]		49	50	14	113
[redacted]	15	38	30	24	107
[redacted]		39	40	6	85
[redacted]	21	14	27	19	81
[redacted]		25	34	9	68
[redacted]		31	24		55
[redacted]		24	21	10	55
[redacted]	1	37	3	6	47
[redacted]		2	9	26	37
[redacted]		6	20	6	32
[redacted]		16	13		29
[redacted]		10	7	5	22
[redacted]		21			21
[redacted]	12	7	2		21
[redacted]		10	6	4	20
[redacted]			16	3	19
[redacted]		18			18
[redacted]			17		17
[redacted]			14		14
[redacted]			1	12	13
[redacted]	5	5	1	1	12
[redacted]				11	11
[redacted]		6	4		10
[redacted]	6	2	2		10
[redacted]			9		9
[redacted]			7		7
[redacted]			5	2	7
[redacted]				6	6
[redacted]		3	3		6
[redacted]			5		5
[redacted]	5				5
[redacted]	4				4
[redacted]			4		4
[redacted]				3	3
[redacted]				2	2
[redacted]			1		1
[redacted]		1			1
[redacted]				1	1
[redacted]			1		1
[redacted]		1			1
Total	76	396	402	230	1104

Figura 68 - Estadísticas: Ficheros por profesor y curso académico

La Figura 69 muestra el número de ficheros subidos por cada asignatura y curso académico en el módulo de *Zona de Descargas*.

Ficheros por asignatura y curso académico

Concepto	2007	2008	2009	2010	Total
Descripción de Lenguajes de Programación [II]		40	44	32	116
Sistemas Hipermedia [MDS]			18	58	76
Informática Aplicada a la Gestión Comercial [LITM]	37	23	16		76
Sistemas Operativos I [ITIS]	1	26	16	18	61
Informática Aplicada a la Gestión de la Empresa [DCE]		36	24		60
Informática de Gestión I [DT]		24	14	21	59
Ofimática [LADE]	17	24	11		52
Programación Dirigida a Objetos [II]		26	2	19	47
Sistemas Operativos II [ITIS]		25	19		44
Sistemas Operativos II [II]		29	14		43
Sistemas Operativos II [ITIG]		19	17		36
Fundamentos del Software [GII]			1	32	33
Ofimática Empresarial [DCE]	7	8	18		33
Sistemas Operativos I [II]		16	11	5	32
Programación Concurrente [II]		14	16		30
Informática de Gestión II [DT]		19	7		26
Análisis y Usos Geográficos de la Información [LG]		12	12		24
Informática Gráfica [II]			23		23
Ingeniería del Software I [II]		16	7		23
Gestión de Redes [IT]		21			21
Asignatura de Prueba [LADED]	10	3	4	1	18
Sistemas Operativos I [ITIG]			9	8	17
Modelado y Visualización de Volúmenes [MDS]			8	7	15
Procesos de Creación Artística, Audiovisual y Multimedia [LBA]		8	6		14
Ampliación de Ingeniería del Software [ITIS]			12		12
Computación Ubicua [MDS]			8	4	12
Digitalización 3D [MDS]			6	6	12
Realidad Virtual [MDS]			11		11
Técnicas Avanzadas de Modelado de Sólidos [MDS]			5	6	11
Diseño de Sistemas Operativos [II]		5	5		10
Metodología de Investigación [MDS]			3	6	9
Ingeniería de la Usabilidad y Ética Informática [MDS]			8		8
Ingeniería Web [MDS]			7		7
Sistemas Informáticos Distribuidos [ITIS]		2	2	3	7
Web semántica [MDS]			5		5
Metodologías y Herramientas para el Desarrollo Evolutivo de Software [MDS]			4		4
Almacenes de Datos y Sistemas OLAP [MDS]			4		4
Fundamentos de Geometría y Geometría Computacional [MDS]				4	4
Técnicas Avanzadas de Modelado de Sistemas de Control y Telecomunicaciones [MDS]			3		3
Curso de Prueba [LADE]	3				3
Sistemas Colaborativos y Procesos de Negocios [MDS]			2		2
Demo Proyecto [II]	1				1
Total	76	396	402	230	1104

Figura 69 - Estadísticas: Ficheros por asignatura y curso académico

La Figura 70 muestra el número de asistencias anotadas por cada profesor y curso académico, mientras que la Figura 71 muestra el número de sesiones de prácticas por asignatura y curso académico. Ambas estadísticas corresponden al módulo de *Control de Asistencia.*

Asistencias anotadas por profesor y curso académico

Concepto	2007	2008	2009	2010	Total
[redacted]	937	1563	1431	364	4295
[redacted]		1734	1387	404	3525
[redacted]	290	945	1169	309	2713
[redacted]	568	937	495	546	2546
[redacted]		345	1493	324	2162
[redacted]	364	1763			2127
[redacted]		254	123	1498	1875
[redacted]	235	305	1270	7	1817
[redacted]		327	533	248	1108
[redacted]			259	783	1042
[redacted]		778	142		920
[redacted]			402	453	855
[redacted]		74	553	208	835
[redacted]			795		795
[redacted]		337	453		790
[redacted]		82	620		702
[redacted]			614		614
[redacted]		554			554
[redacted]				500	500
[redacted]	24	84	292	30	430
[redacted]				396	396
[redacted]	390				390
[redacted]		151		217	368
[redacted]		359			359
[redacted]		195		150	345
[redacted]			214		214
[redacted]			158		158
[redacted]		145			145
[redacted]				92	92
[redacted]				55	55
[redacted]	20	5	2		27
[redacted]		15			15
[redacted]				7	7
[redacted]		3			3
Total	2828	10955	12405	6591	32779

Figura 70 – Estadísticas: Asistencias anotadas por profesor y curso académico

Sesiones de prácticas por asignatura y curso académico

Concepto	2007	2008	2009	2010	Total
Informática Aplicada a la Gestión de la Empresa [DCE]		149	136		285
Ofimática [LADE]	67	70	73		210
Informática de Gestión I [DT]		54	69	52	175
Sistemas Operativos I [ITIS]		48	68	20	136
Sistemas Operativos II [ITIS]		65	54		119
Programación Dirigida a Objetos [II]		46	50	16	112
Fundamentos del Software [GII]				110	110
Sistemas Operativos I [ITIG]			47	44	91
Informática Aplicada a la Gestión Comercial [LITM]	26	31	33		90
Ofimática Empresarial [DCE]	22	26	24		72
Sistemas Operativos I [II]			22	42	64
Programación Concurrente [II]		36	20		56
Sistemas Operativos II [II]		17	35		52
Análisis y Usos Geográficos de la Información [LG]		41	9		50
Ampliación de Ingeniería del Software [ITIS]			33		33
Ingeniería del Software I [II]		22		6	28
Sistemas Operativos II [ITIG]		20	5		25
Sistemas Informáticos Distribuidos [ITIS]		10	9		19
Asignatura de Prueba [LADED]	4	9	3	3	19
Informática de Gestión II [DT]		12			12
Metodologías y Herramientas para el Desarrollo Evolutivo de Software [MDS]			10		10
Sistemas Hipermedia [MDS]				5	5
Curso de Prueba [LADE]	5				5
Demo Proyecto [II]	2				2
Fundamentos de Geometría y Geometría Computacional [MDS]				2	2
Diseño de Sistemas Operativos [II]		1			1
Procesos de Creación Artística, Audiovisual y Multimedia [LBA]		1			1
Total	126	658	700	300	1784

Figura 71 - Estadísticas: Sesiones de prácticas por asignatura y curso académico

Capítulo 6. Mantenimiento

Existen una serie de tareas asociadas al trabajo que he estado desempeñando durante la realización de este proyecto que no están relacionadas con el análisis, diseño o implementación de funcionalidades, sino que se centran en el mantenimiento del servidor que hospeda a Tutor.

Si bien estas tareas pueden ser, por lo general, rutinarias, no por ello son despreciables, pues dado que Tutor es un sistema en producción, se hace necesario realizar un seguimiento constante del servidor. Asimismo, la realización de estas tareas requiere conocer en profundidad cada uno de los componentes que conforman el servidor Tutor, desde el *hardware* de la máquina hasta las versiones de cada una de las aplicaciones instaladas en ella, pasando por otros elementos, como pueden ser el SAI que protege al equipo.

También se hace necesario en la realización de estas tareas la capacidad de poder realizar análisis de los diferentes informes y *logs* que proporciona el sistema.

6.1. Chequeo de logs de Apache

Apache hace uso de varios ficheros de registro para reflejar su estado y los diferentes sucesos producidos. Los dos ficheros más importantes a este respecto son *error.log* y *sslerror.log*. El primero de ellos muestra los errores producidos en el servidor. El segundo también muestra los errores, pero en este caso sólo muestra los errores producidos bajo una conexión segura.

6.1.1 Fichero error.log

El fichero *error.log* almacena los eventos relacionados con los inicios y paradas del servidor, los errores producidos en alguno de los módulos que componen Apache, las solicitudes de acceso a ficheros que no existen en el servidor,… Un ejemplo de extracto de dicho fichero se muestra en la Figura 72.

En el caso de Tutor es muy típico encontrar intentos de acceso indebido, tal como se puede ver en la Figura 72. En general, estos intentos de acceso suelen ser realizados por *bots* que se dedican a rastrear Internet en busca de servidores que presenten vulnerabilidades. En la Figura 72 se puede ver como el 25 de Diciembre hubo un rastreo de este tipo de vulnerabilidades, pues se intentó acceder a carpetas con nombres tales como *admin/*, *mysql/*, *mysqladmin/*, *dbadmin/*. Para paliar este tipo de ataques, la carpeta que contiene el gestor de base de datos de Tutor se encuentra bajo un nombre no genérico.

Si bien es cierto que la mejor medida de protección a este respecto sería hacer inaccesible esta carpeta desde el exterior, existen situaciones en las que se requiere poder acceder a la base de datos de Tutor de forma remota, como por ejemplo, cuando la utilidad de Conexión a Escritorio Remoto no está en la máquina cliente, cuando hay un usuario operando de forma local en el servidor, cuando ya hay otro usuario conectado de forma remota al servidor, etc. De cualquier manera, hasta el momento no se ha dado el caso de que un ataque de este tipo haya afectado a Tutor.

```
129299 [Fri Dec 24 16:32:43 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129300 [Fri Dec 24 16:32:44 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129301 [Fri Dec 24 16:32:44 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129302 [Fri Dec 24 16:32:44 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129303 [Fri Dec 24 16:32:45 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129304 [Fri Dec 24 16:32:45 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129305 [Fri Dec 24 16:32:46 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129306 [Fri Dec 24 16:32:46 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129307 [Fri Dec 24 16:32:46 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129308 [Fri Dec 24 16:32:47 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129309 [Fri Dec 24 16:32:47 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129310 [Fri Dec 24 16:32:48 2010] [error] [client 207.182.98.19] File does not exist: C:/xampp/xampp/htdocs/db
129311 [Fri Dec 24 17:06:00 2010] [error] [client 66.249.71.198] File does not exist: C:/xampp/xampp/htdocs/robots.txt
129312 [Fri Dec 24 17:06:00 2010] [error] [client 66.249.71.198] File does not exist: C:/xampp/xampp/htdocs/dss-calculator
129313 [Fri Dec 24 19:18:17 2010] [error] [client 65.55.3.135] File does not exist: C:/xampp/xampp/htdocs/robots.txt
129314 [Fri Dec 24 21:48:48 2010] [error] [client 66.249.72.140] File does not exist: C:/xampp/xampp/htdocs/robots.txt
129315 [Fri Dec 24 22:16:37 2010] [error] [client 207.46.204.231] File does not exist: C:/xampp/xampp/htdocs/robots.txt
129316 [Sat Dec 25 00:44:40 2010] [error] [client 207.46.199.178] File does not exist: C:/xampp/xampp/htdocs/robots.txt
129317 [Sat Dec 25 09:49:01 2010] [error] [client 66.249.72.140] File does not exist: C:/xampp/xampp/htdocs/robots.txt
129318 [Sat Dec 25 10:11:52 2010] [error] [client 208.103.230.241] File does not exist: C:/xampp/xampp/htdocs/sqlitemanager
129319 [Sat Dec 25 11:08:10 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/scripts
129320 [Sat Dec 25 11:08:11 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/admin
129321 [Sat Dec 25 11:08:12 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/admin
129322 [Sat Dec 25 11:08:12 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/admin
129323 [Sat Dec 25 11:08:13 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/db
129324 [Sat Dec 25 11:08:14 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/dbadmin
129325 [Sat Dec 25 11:08:14 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/myadmin
129326 [Sat Dec 25 11:08:15 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/mysql
129327 [Sat Dec 25 11:08:16 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/mysqladmin
129328 [Sat Dec 25 11:08:16 2010] [error] [client 202.216.241.194] File does not exist: C:/xampp/xampp/htdocs/typo3
```

Figura 72 - Extracto del fichero error.log de Apache

6.1.2 Fichero sslerror.log

Como se ha comentado previamente, el fichero *sslerror.log* almacena los errores producidos bajo conexión segura, mostrándose en la Figura 73 un ejemplo de extracto de este fichero. Dado el entorno de producción en el que se encuentra Tutor, la configuración de PHP está pensada para que vuelque los errores de código que se puedan producir en la página a este fichero, en vez de ser mostrados por pantalla, como sería lógico en un entorno de desarrollo. Por ello, se hace necesario revisiones esporádicas del fichero para comprobar si alguna funcionalidad está generando errores, o si hay algún comportamiento inesperado. Al igual que en el fichero *error.log*, este fichero refleja los intentos de acceso a carpetas no existentes o restringidas. Es interesante ver que en ocasiones hay intentos de acceso a la carpeta de plantillas de Tutor, la cual es de acceso restringido. Una posible explicación a este hecho podría ser el intento de buscar vulnerabilidades en Tutor a través del análisis de las plantillas existentes en el sistema. En cualquier caso, hasta la fecha no se conoce que ningún ataque haya tenido éxito.

```
413 [Sat Dec 25 18:33:12 2010] [error] [client 66.249.72.205] File does not exist: C:/xampp/xampp/htdocs/public/templates/
414 [Sat Dec 25 20:52:28 2010] [error] [client 66.249.72.205] File does not exist: C:/xampp/xampp/htdocs/public/contact/fu
415 [Sat Dec 25 21:29:12 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
416 [Sat Dec 25 21:29:12 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
417 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
418 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
419 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
420 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
421 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
422 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
423 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
424 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
425 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
426 [Sat Dec 25 21:29:13 2010] [error] [client 87.222.163.162] PHP Warning:  session_destroy() [<a href='function.session-
427 [Sat Dec 25 23:14:42 2010] [error] [client 66.249.72.205] File does not exist: C:/xampp/xampp/htdocs/public/academic
428 [Sun Dec 26 02:10:27 2010] [error] [client 66.249.72.205] File does not exist: C:/xampp/xampp/htdocs/public/nuevotutor
429 [Sun Dec 26 09:48:42 2010] [error] [client 66.249.72.18] Directory index forbidden by rule: C:/xampp/xampp/htdocs/publ
```

Figura 73 - Extracto del fichero sslerror.log de Apache

6.2. Chequeo del sistema operativo

En la actualidad, Tutor corre sobre un sistema operativo Windows. Este sistema proporciona, como herramienta de control, el *Visor de sucesos*, donde se almacenan todos los eventos relacionados con servicios, fallos del sistema, reinicios inesperados,... Si bien, en general, este fichero no suele presentar demasiadas anomalías, es útil para detectar si se están produciendo fallos en las copias de seguridad, o tras una caída del servidor. En la Figura 74 se pueden ver, entre otros, los eventos de información producidos por el proceso de copia de seguridad (NTBackup), así como por la parada y reinicio del servidor MySQL, que se produce al inicio de cada mes para generar una copia de seguridad integral del sistema.

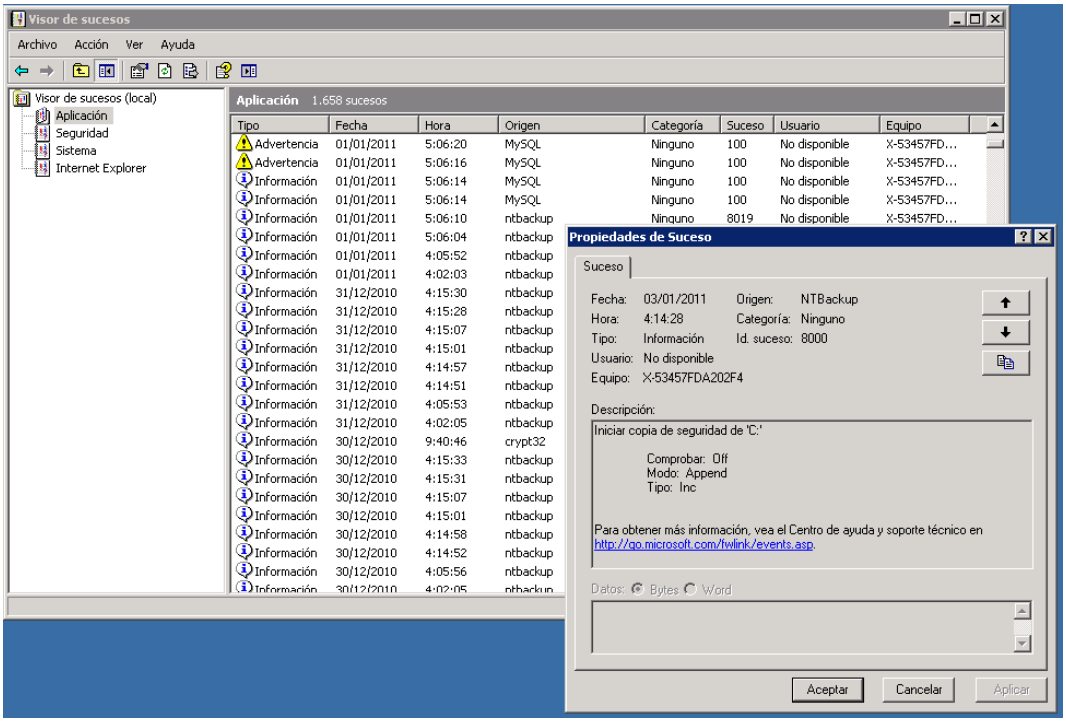


Figura 74 - Captura del visor de sucesos de Windows (1)

A lo largo del curso académico 2010/11, se activó la compresión en tiempo real de las páginas generadas por Tutor, con el objetivo de reducir el tamaño de los datos a transferir entre el servidor y los clientes. La activación de esta opción supuso un deterioro en la calidad del sistema, pues, debido a problemas en la gestión de memoria de esta opción, el sistema agotaba los recursos disponibles en el servidor durante las horas de mayor acceso.

Gracias al registro de eventos, tal y como se aprecia en la Figura 75, se pudo aislar y solucionar la causa del problema. En esta figura pueden verse los errores producidos en Apache durante la mañana del martes 9 de noviembre.

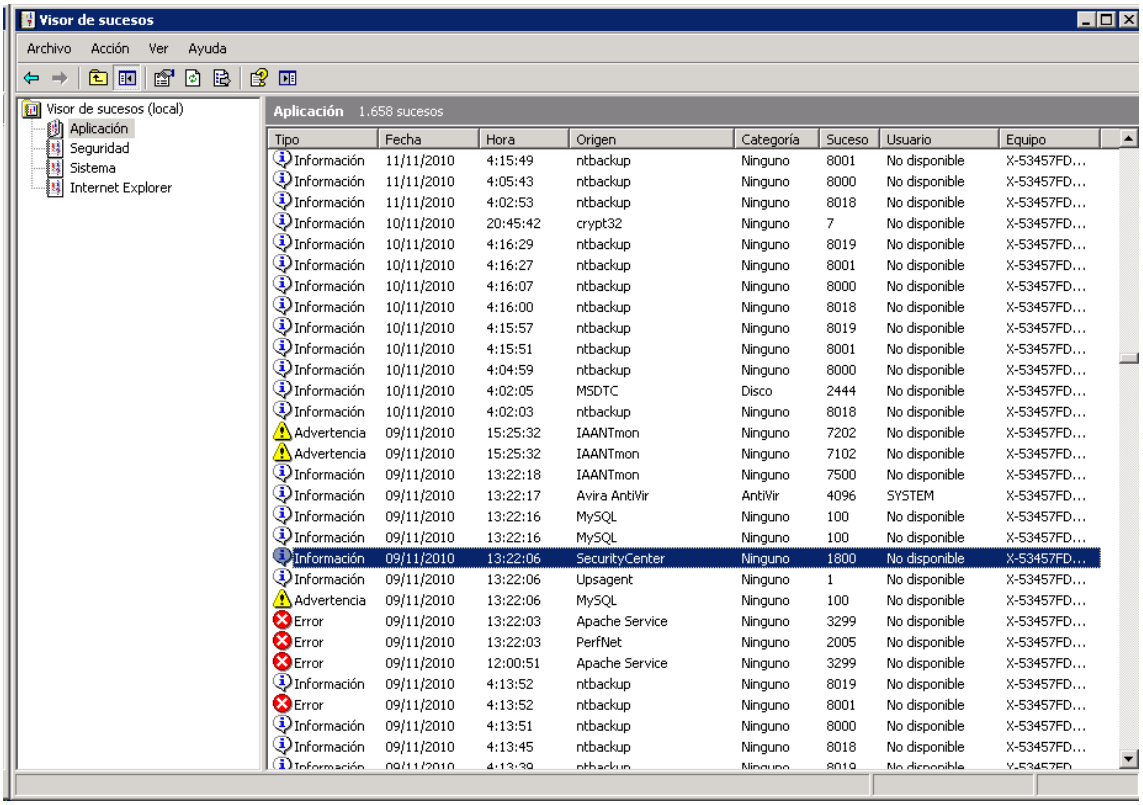


Figura 75 - Captura del visor de sucesos de Windows (2)

Por otra parte, otra tarea relacionada con la supervisión del sistema operativo consiste en instalar las actualizaciones de seguridad que suelen ir apareciendo para el mismo. Si bien esta tarea no requiere de mayor complicación, en ocasiones requieren de un reinicio del servidor. Por ello, resulta necesario realizar estas actualizaciones en periodos de poca demanda del servidor, tales como fines de semana o madrugadas. También es necesario realizar una comprobación del sistema tras la instalación de una actualización, con el objetivo de verificar que todo sigue funcionando correctamente.

6.3. Copias de seguridad

Tutor mantiene en la actualidad un triple sistema de copias de seguridad.

El primer sistema consiste en una solución hardware basada en un RAID 1, o de espejo, entre los dos discos duros del sistema. El correcto funcionamiento de este RAID se comprueba mediante la utilidad *Intel Matrix Storage Console* (Figura 76). Hasta la fecha, no se ha producido ningún error derivado del mal funcionamiento de los discos duros o del RAID.

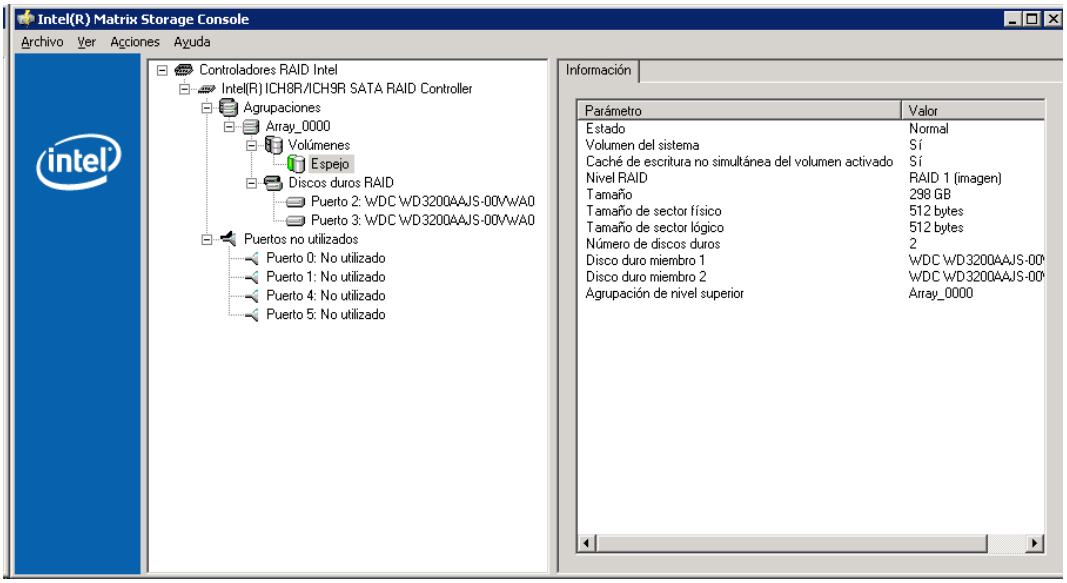


Figura 76 - Utilidad para la supervisión del RAID de Tutor

El segundo sistema consiste en una copia de seguridad en el propio disco duro de Tutor. Este proceso se realiza mediante un script que lanza diariamente a las 4 de la mañana la utilidad *NTBackup*, generando una copia de seguridad incremental del servidor Apache, de los datos del sistema (ficheros de asignaturas, trabajos de alumnos,…) y de la base de datos. Cada día 1 del mes en curso se genera una nueva copia íntegra del sistema. En la actualidad se mantienen las copias correspondientes a los últimos 4 meses de funcionamiento. El correcto funcionamiento de esta copia de seguridad se puede ver mediante el visor de sucesos de Windows (Sección 6.2) o mediante un fichero de texto generado por el script, Figura 77.

Mediante este sistema se pueden recuperar de forma fácil trabajos, ficheros, anotaciones de asistencias, etc., de los últimos 4 meses. Durante el periodo en el que ha transcurrido el trabajo realizado en este proyecto, se ha hecho uso de esta copia de seguridad en dos ocasiones, con el objetivo de recuperar anotaciones realizadas en el control de asistencia, perdidas por equivocación tras haber sido sobrescritas.

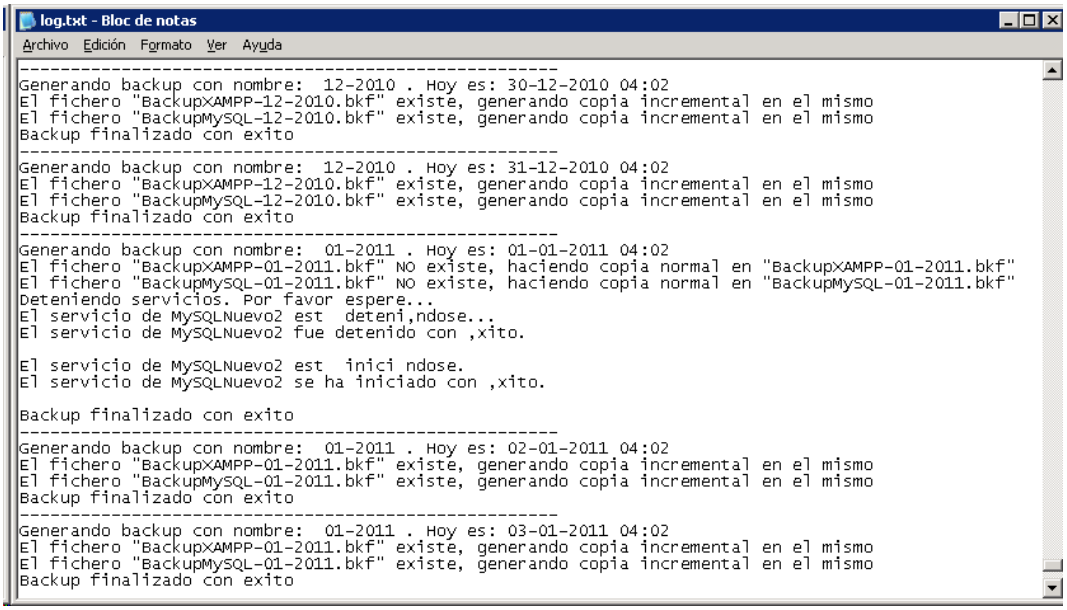


Figura 77 - Fichero de registro para las copias de seguridad de Tutor

Por último, el tercer sistema de copia de seguridad consiste en guardar en una copia exacta de las carpetas de Apache y MySQL en un repositorio SVN externo al servidor, tarea que se ejecuta diariamente. De esta forma, se garantiza que, ante la pérdida total del servidor, exista una copia alojada en otra máquina, de modo que permita poner de nuevo en funcionamiento el sistema rápidamente, aún en el caso de tener que hacerlo en otro servidor.

6.4. Asistencia al usuario

Una vez más, el hecho de ser Tutor un sistema en producción utilizado diariamente por bastantes usuarios conlleva que dicho uso derive en la detección de errores o problemas, en propuestas de nuevas funcionalidades, en peticiones de ayuda,… por parte de los usuarios del mismo. En cualquiera de estos casos, se requiere dar una rápida respuesta al usuario, con el objetivo de mantener un grado alto de confiabilidad en el sistema.

El catálogo de notificación de errores y de planteamiento de consultas y sugerencias derivados del uso de Tutor es de diversa índole, desde estudiantes que no recuerdan su contraseña a estudiantes que no aparecen registrados en el sistema, pasando por errores de funcionamiento en funcionalidades, o alumnos que por algún motivo tienen problemas para entregar un trabajo, por poner algunos ejemplos. Del mismo modo, las soluciones pueden variar desde el remitir al usuario a su correspondiente profesor de teoría o prácticas, hasta el análisis detallado de los datos ofrecidos por el profesor o alumno con el objetivo de hallar solución al problema descrito.

En el caso de las solicitudes de nuevas funcionalidades (fundamentalmente por parte de profesores), en función de la complejidad de éstas, se actúa de diferente manera. Si la inclusión en el sistema de la funcionalidad propuesta no conlleva gran complejidad, tanto

técnica como a nivel de esfuerzo y tiempo necesario, se intenta implementar a la mayor brevedad posible, con el objetivo de mostrar al usuario que su *feedback* es realmente tenido en cuenta para el desarrollo de Tutor. En caso contrario, si la propuesta concierne cierta complejidad, se anota en la lista de posibles extensiones del sistema, con el fin de considerar su desarrollo en el futuro.

A título personal, el número de cadenas de mensajes a las que he tenido que responder, previa realización de la acción correspondiente en muchos casos, desde el 1 de septiembre de 2010 al 21 de diciembre de 2010, ha sido de aproximadamente 80.

Capítulo 7. Conclusiones y trabajo futuro

7.1. Conclusiones

Si algo quiero destacar de esta memoria es que Tutor es un proyecto *vivo*, vivo en todos los aspectos.

A diferencia de otros proyectos Tutor es un sistema que está funcionando, con los problemas que eso conlleva. Cualquier cambio, mejora o arreglo es auditado por más de 100 usuarios a la hora. Esto supone una responsabilidad mayor, pues cualquier fallo en el sistema, cualquier caída del servidor o cualquier otro imprevisto implica dejar de dar servicio o poner en compromiso los datos y el trabajo de todos los usuarios de la plataforma.

Así, a las tareas de desarrollo de un proyecto convencional se le ha sumado en mi caso las tareas de mantenimiento, seguridad, supervisión, asistencia al usuario,… de un sistema que realmente está en producción. Esta dedicación extra implica un grado de compromiso y exigencia que es difícilmente plasmable en una memoria, pero cuyo trabajo está ahí. Afortunadamente, hasta la fecha, no se ha producido ninguna pérdida de datos grave en Tutor, y las pocas caídas del sistema que ha habido han sido debidas en su mayoría a problemas eléctricos en el lugar donde se ubica el servidor. Del mismo modo, los problemas surgidos y notificados por profesores y estudiantes se han solucionado, en la mayoría de los casos, en un corto espacio de tiempo, inferior al par de horas en muchas ocasiones.

Sin embargo, hay un hecho que quiero recalcar al respecto de todo esto: me gusta esta responsabilidad. No solo me gusta, sino que agradezco la posibilidad que se me ha brindado y se me brinda de, como ya he mencionado antes, trabajar en un sistema *vivo*.

Si hay algo que he tenido claro a lo largo de la carrera es que me gustan las cosas prácticas, y útiles. Extrapolar esto al proyecto fin de carrera me implicaba, dado que iba a emplear gran cantidad de tiempo en el mismo, intentar realizar algo a lo que se le diese uso realmente, o que no cayese en saco roto. Dicho fríamente, no quería que mi trabajo acabase adornando la estantería de algún despacho.

Existe un tema recurrente al respecto Tutor que podría obviar, la eterna pregunta del "¿Y por qué una plataforma nueva? ¡Si hay miles!". Sería incongruente decir que me considero práctico y al mismo tiempo negar que me haya hecho esta pregunta alguna vez. Y, aunque a veces he titubeado, hay dos argumentos que me ayudan a responder a esta pregunta. El primero de ellos es que Tutor está diseñado en gran medida por y para sus usuarios finales. Hace mucho tiempo que perdí la cuenta de cuantas horas de Skype puedo haber pasado hablando con mi coordinador, Miguel Hornos, al respecto de mejoras, cambios, aspectos de

interfaz, problemas detectados, funcionalidades futuras, seguridad, y otros muchos aspectos de Tutor. Sin embargo, este primer argumento no responde al hecho de que todo el trabajo realizado se podría haber hecho mediante módulos en plataformas ya existentes. Para esta pregunta, mi argumento es que en Tutor siempre he intentado hacer uso de tecnologías y patrones de diseño modernos, más propios de la Web 2.0, como puede ser, por ejemplo, el uso de patrones AJAX.

Este hecho implica el estar descubriendo, probando e investigando nuevas técnicas casi a diario. Facebook, probablemente la página más usada del mundo en la actualidad junto a Google, ha cambiado su interfaz general 2 veces durante los últimos 5 meses. Google renovó hace poco su famoso buscador, añadiéndole Google Instant, claro ejemplo de uso de tecnologías AJAX. El eterno renovarse o morir.

Así, a día de hoy, un proyecto que en cierto modo consiste en transformar la estructura de un sistema web y dotarle de nuevas funcionalidades, labor que probablemente se podría resolver fácilmente con tan sólo el uso de PHP+HTML+CSS, se transforma en un estudio constante de hacia dónde van las tendencias de uso e interfaces web, cómo evolucionan las librerías de código libre, qué recursos existentes se pueden reaprovechar,… JQuery, Prototype, YUI, Smarty, HTML5, JSON, GWT, Jabber, Openfire,… son sólo una pequeña lista de herramientas, lenguajes, protocolos o tecnologías que he conocido y utilizado a lo largo del tiempo que he empleado en este proyecto.

Por otra parte, el tener la responsabilidad de renovar la estructura interna de Tutor y el haber adquirido una serie de hábitos a la hora de diseñar sus módulos implica transmitir y discutir esta estructura y/o estos hábitos con otros desarrolladores de Tutor. Durante el periodo en el que ha sido desarrollado este proyecto he compartido tiempo con otros 3 desarrolladores. El debatir sobre cómo resolver problemas, el enseñarles lo que sabía o ayudarles al respecto de situaciones que yo ya me había encontrado previamente ha sido otro de los retos de este proyecto. Ser “auditado” por 100 usuarios a la hora es complejo. Ser “auditado” por otros 3 desarrolladores lo complica aún más.

El tema de cuál es el futuro de Tutor se trata más detalladamente en la sección 7.2. Pero resumidamente se puede decir que en Tutor queda tanto trabajo por hacer como funcionalidades se le quieran añadir en el futuro. Una vez que se le ha dotado de modularidad, la tarea de añadir o quitar nuevas opciones se hace más sencilla. Y de cualquier forma, aún en el caso de que se decidiese no desarrollar más funcionalidades para el sistema en el futuro, las tareas de mantenimiento descritas en el capítulo 6 seguirían estando presentes.

A título personal, ya he reutilizado partes de Tutor en otros proyectos, tales como la clase *Template* o el *Gestor de Módulos*. Y dado que Tutor es software libre, no descarto, en un futuro, realizar las modificaciones necesarias para que pueda ser usado en otro tipo de centros, tales como colegios o institutos y, por qué no, intentar obtener una rentabilidad económica con la prestación de servicios asociada al sistema, tales como instalación, adaptación, mantenimiento, etc.

Como conclusión última diré que, a lo largo del proyecto, he adquirido un vínculo o compromiso personal con Tutor. Vínculo o compromiso que, si bien es cierto, perdurará en el

tiempo, toma un carácter de punto y aparte al término de este proyecto. A día de hoy no sé cómo será dicho punto y aparte, pero si algo tengo claro es que no será fácil llegar al día en el que, en la medida de lo posible, deje de contribuir con Tutor.

7.2. Trabajo futuro

Para hablar de trabajo futuro desde la perspectiva de esta memoria hay que, en primer lugar, recordar el título que da nombre a la misma: "Hacia Tutor 2.0: Aplicando características de la Web 2.0 a Tutor". Recordado este título, la pregunta que cabe formularse es: ¿Se han aplicado estas características? En mi opinión, sí. Ahora bien, la segunda pregunta que cabría formularse es: ¿Se han aplicado *lo suficiente*? En este caso, mi respuesta sería no. Pero eso no significa que no se hayan sentado las bases para aplicarlas en un futuro, pues la remodelación del sistema que se ha desarrollado en el Capítulo 3, pese a no ser vistosa de cara a nuevas funcionalidades, sí que ayuda enormemente a futuros desarrollos sobre la plataforma.

A este respecto, habría que pararse a pensar qué es lo que se busca con el "acercamiento a la web 2.0". Tutor aún no tiene un gran carácter social. De hecho, hasta donde conozco, no existe una plataforma educativa que tenga un gran carácter social. Existen redes sociales educativas, donde los profesores pueden intercambiar material, los alumnos pueden consultarlo, o compartirlo también, redes donde sus miembros pueden comunicarse entre ellos. Salvando las distancias, Tutor no está lejos de este tipo de uso.

Decir que Tutor está a la altura de Moodle sería poco menos que engañarse. Pero, en mi opinión, también sería engañarse decir que Moodle tiene un gran carácter social. Existen soluciones que integran Moodle con Elgg [7] y le dotan de carácter social. También existen soluciones como Mahara [32], que combina un gestor de contenidos para portafolios electrónico con una red social. Y aun así, sigo pensando que estas soluciones no tienen un gran carácter social.

Tiene una explicación. Desde mi punto de vista, la clave de un gran carácter social es que los estudiantes pueden estar al corriente de lo que están haciendo los demás estudiantes. Y es aquí donde surge el debate. Volviendo a poner en el punto de mira en Facebook, una de las claves de su éxito, a mi juicio, es que nos permite ver cómo se relacionan nuestros amigos entre ellos. Nos permite ver, siempre que lo permitamos, quién se hizo fotos con quien el día anterior, quién ha quedado con quien para mañana, quién va a ir al evento propuesto por alguien,…

La pregunta es, ¿sería disparatado aplicar este enfoque a una plataforma orientada a la enseñanza? Imaginemos por un momento un sistema en el que se producen situaciones como las siguientes:

Tres alumnos, Juan, Laura y Ana, empiezan a desarrollar un trabajo sobre la propia plataforma. Automáticamente un mensaje les aparece al resto de compañeros en su *Tablón de*

actividad (página inicial de la plataforma donde ven lo que está pasando, algo similar al muro de Facebook):

"Juan, Laura y Ana han empezado a desarrollar el trabajo sobre Economía Mundial de la Asignatura de Economía 1".

Este mensaje podría ser comentado por sus compañeros. Y podríamos encontrar comentarios del siguiente tipo:

- Carlos: *"¿Ya vais a empezar? ¡Si es para dentro de dos semanas!"*
- Andrés: *"Pff… bueno, ya que empezáis vosotros, me pongo yo también, y así nos vamos ayudando con las dudas"*
- Profesora Miriam: *"¡No dejéis el trabajo para el último día, que luego ya sabéis lo que os pasa!"*
- Juan: *"Profesora, ¿podrías echarle un vistazo a lo que llevo de trabajo? Tengo algunas dudas al respecto. ¡Gracias!"*
- María: *"Juan… ¡eres un pelota! :D"*

Este podría ser un ejemplo de gran carácter social. Y, como este ejemplo, podríamos hablar de otros muchos, como el de un profesor que sube un archivo y los alumnos le comentan que no entienden una página del mismo, o un alumno que publica un video y el profesor y resto de estudiantes dan su valoración al respecto.

Podemos ir aun más allá y pensar en un sistema donde los alumnos desarrollen un trabajo, cuya evaluación y comentarios al respecto pueda ser realizado por el resto de alumnos de la clase, con la supervisión del profesor correspondiente. Me reitero, una vez más, para decir que las posibilidades son innumerables.

No obstante, todo este planteamiento conlleva el polémico factor de la privacidad e individualidad del trabajo. En cierta medida, la entrada en vigor del Plan Bolonia llama a fomentar el trabajo en equipo y la colaboración entre estudiantes. Sin embargo, ¿tiene sentido que un alumno tenga forzosamente que indicar a sus compañeros lo que está haciendo o dejando de hacer académicamente? El sentido lógico dice que no. Pero, por otra parte, es un hecho que gran parte de los estudiantes universitarios se reúnen y comparten su tiempo para, por ejemplo, resolver y comparar prácticas, independientemente de que éstas sean individuales o no. Y también es un hecho que existe gran cantidad de gente que "se va a la biblioteca a estudiar porque en casa solo no puede concentrarse", lo que a priori puede parecer un contrasentido. Sin embargo, hay personas a las que el estudiar rodeado de gente le supone menor esfuerzo que estudiar solo, por motivos tales como saber que no es la única persona en esa situación, o que en un momento dado puede consultar las dudas que le surjan a algún compañero que esté a su alrededor. Sin entrar en debates filosóficos, se suele considerar al ser humano sociable por naturaleza, independientemente del ámbito del que hablemos.

Recapitulando, mi opinión en cuanto a trabajo futuro en Tutor es que existen dos caminos que se pueden tomar a la finalización de este proyecto:

- El primero de ellos es continuar reinventando la rueda e implementar funcionalidades que, si bien contemplan ciertas variaciones, podrían implementarse o adaptarse en otras plataformas ya existentes.
- El segundo camino es el camino arriesgado, el camino de darle un enfoque a Tutor que realmente lo diferencie de otras plataformas educativas existentes. Es un camino que implica un gran debate en cuanto a la individualidad de la realización de actividades académicas concierne y que conlleva una implicación a nivel docente por parte de los profesores que estuviesen dispuestos a experimentar ese camino. Y, como camino experimental y arriesgado, puede ser todo un éxito o todo un fracaso, pero en cualquiera de los dos casos acabaría con el tópico de "una plataforma educativa más".

Si se optara por este segundo camino, y una vez superadas las cuestiones éticas en cuanto a la información que puede ser compartida, habría que pensar en funcionalidades concretas. El primer paso, sin duda, sería desarrollar lo que anteriormente he citado como *Tablón de actividad*. Y, una vez desarrollado éste, deberían adaptarse los módulos existentes, con el objetivo de que el *Tablón de actividad* reflejase los eventos del sistema y los usuarios pudieran interaccionar entre ellos.

A partir de ahí, habría que estudiar los modelos de interacción social que están teniendo éxito actualmente en Internet, diseñando a continuación modelos de interacción social que sean aplicables académicamente, innovando por tanto en este aspecto. Un dato anecdótico de este proyecto es que, pese a que se partía de que la funcionalidad de los *Foros* y de la *Mensajería Interna* era diferente, estudiantes del sistema están usando la *Mensajería Interna* para enviar mensajes a todos sus compañeros. Este tipo de uso ya era posible hacerlo mediante la funcionalidad de foros, disponible incluso antes de empezar el desarrollo de este proyecto. Sin embargo, mientras que el número total de hebras en foros durante los últimos 15 meses es de 24, en tan sólo 3 meses el número de cadenas de mensajes que involucran a, al menos, un profesor y un grupo, es de 60. Mientras que el número de cadenas de mensajes que sólo involucran grupos, sin sus profesores, es de 25. Mi lectura de estos datos es la de que el usuario medio de Tutor está más acostumbrado a hacer uso de sistemas de mensajería, en lugar de foros, pues es el modelo que siguen actualmente las principales plataformas sociales. No obstante, considero que los datos a este respecto no son concluyentes y que pueden existir otras lecturas.

En cualquier caso, creo que queda claro cuál sería mi enfoque ideal para el futuro de Tutor. No puedo aventurarme a decir si tendría éxito o no, pero indudablemente no sería un reto si el éxito estuviese asegurado de antemano.

Anexos

A. Estructura de las tablas usadas en Mensajería interna

El diseño del módulo de *Mensajería Interna,* en cuanto a base de datos se refiere, está compuesto por 4 tablas: *pm_threads*, *pm_threads_receivers*, *pm_messages* y *pm_messages_stats.* A continuación se muestra la estructura y una breve descripción de cada una de estas tablas.

Tabla pm_threads

Esta tabla, cuya estructura se muestra en la Figura 78, almacena las cadenas de mensajes, y está formada por 5 campos:

- **id**: Identificador único de la cadena de mensajes.
- **subject**: Asignatura a la que pertenece la cadena de mensajes.
- **title**: Título de la cadena de mensajes.
- **parent_thread**: Cadena de mensajes padre, si la hubiera, de la que proviene esta nueva cadena. Se usa cuando se ha respondido individualmente a una cadena de mensajes entre múltiples usuarios.
- **creator**: Identificador del usuario que crea la cadena de mensajes.

Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
id	int(10)		UNSIGNED	No		auto_increment
subject	int(10)		UNSIGNED	No		
title	varchar(200)	utf8_unicode_ci		No		
parent_thread	int(10)		UNSIGNED	Sí	*NULL*	
creator	int(10)		UNSIGNED	No		

Figura 78 - Estructura de la tabla pm_threads

Tabla pm_threads_receivers

Esta tabla, cuya estructura se muestra en la Figura 79, almacena la relación entre cada cadena de mensajes y los receptores de la misma, y está formada por 3 campos:

- **idthread**: Identificador de la cadena de mensajes de la asociación.
- **kind**: Tipo de receptor, *student*, *teacher* o *group*.
- **idreceiver**: Identificador del receptor.

Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
idthread	int(10)			No		
kind	varchar(10)	utf8_unicode_ci		No		
idreceiver	int(10)			No		

Figura 79 - Estructura de la tabla pm_threads_receivers

Tabla pm_messages

Esta tabla, cuya estructura se muestra en la Figura 80, almacena los mensajes, y está formada por 5 campos:

- **id**: Identificador único del mensaje.
- **idthread**: Identificador de la cadena de mensajes a la que pertenece.
- **idauthor**: Identificador del autor del mensaje.
- **text**: Texto del mensaje.
- **datetime**: Fecha y hora del envío del mensaje.

Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
id	int(10)		UNSIGNED	No		auto_increment
idthread	int(10)		UNSIGNED	Sí	*NULL*	
idauthor	int(10)		UNSIGNED	No		
text	text	utf8_unicode_ci		No		
datetime	datetime			No		

Figura 80 - Estructura de la tabla pm_messages

Tabla pm_messages_status

Esta tabla, cuya estructura se muestra en la Figura 81, almacena la relación existente entre cada mensaje y usuario, asociándole un estado a la misma, y está formada por 3 campos:

- **idmessage**: Identificador del mensaje.
- **iduser**: Identificador del usuario.
- **status**: Estado de la asociación. Tal y como se explicó en el apartado 3.4.1, los valores pueden ser: *mine*, *unread*, *read* y *deleted.*

Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
idmessage	int(10)		UNSIGNED	No		
iduser	int(10)		UNSIGNED	No		
status	varchar(10)	utf8_unicode_ci		No		

Figura 81 - Estructura de la tabla pm_messages_status

B. Contenido del CD

Junto a esta memoria se entrega un CD, cuyo contenido, distribuido en sus correspondientes carpetas es el siguiente:

Código fuente

Dentro de la carpeta de código fuente se encuentra una copia del mismo, junto a los scripts de instalación del sistema, la estructura de la base de datos, y una serie de datos de ejemplo. La instalación de una copia de Tutor requiere tener en funcionamiento un servidor Apache y un servidor MySQL.

El proceso de instalación es sumamente fácil, basta volcar el contenido de la carpeta código fuente a una carpeta del servidor y acceder a través del navegador a esta carpeta. Un script de instalación solicitará al usuario los datos correspondientes a la base de datos y al servidor donde será instalado Tutor antes completar la instalación.

En cualquier caso, dado el carácter de estar en constante actualización, se recomienda al usuario interesado en el código de Tutor que obtenga una copia del mismo mediante la versión distribuida en la forja de Red Iris [12].

Documentación

Esta memoria se incluye en el CD en formato PDF. La versión electrónica del documento tiene debidamente enlazadas todas las referencias internas y externas usadas a lo largo del mismo, facilitando la navegación a través del contenido.

Bibliografía

[1] AJAX, Wikipedia, http://es.wikipedia.org/wiki/AJAX
[2] Alexa: The Web information company, http://www.alexa.com
[3] Baidu, http://www.baidu.com
[4] Blogger, http://www.blogger.com
[5] Código spaghetti, Wikipedia, http://es.wikipedia.org/wiki/Código_spaghetti
[6] ejabberd: The Erlang Jabber/XMPP Daemon, http://www.ejabberd.im/
[7] Elgg: Social Networking Engine, http://www.elgg.org/
[8] Facebook Developers, "Facebook: Chat reaches 1 billion messages", Facebook Engineering's notes, http://www.facebook.com/note.php?note_id=91351698919
[9] Facebook, http://www.facebook.com
[10] Firebug, http://getfirebug.com/
[11] Flex and Jabber, http://flashflex.com/flex-and-jabber/
[12] Forja de Tutor, Red Iris, https://forja.rediris.es/projects/cusl3-tutor/
[13] Free Software Foundation, General Public License Version 3, GNU.org http://www.gnu.org/licenses/gpl-3.0.html
[14] General Public License, Wikipedia, http://es.wikipedia.org/wiki/GNU_General_Public_License
[15] Google Talk, http://en.wikipedia.org/wiki/Google_Talk
[16] Google, http://www.google.es
[17] GWT: Google Webmaster Toolkit, http://code.google.com/intl/es-ES/webtoolkit/
[18] ICQ, Wikipedia, http://es.wikipedia.org/wiki/ICQ
[19] IFrame Call, Ajax Patterns , http://ajaxpatterns.org/IFrame_Call
[20] IFrame Tag, W3Schools http://www.w3schools.com/tags/tag_iframe.asp
[21] Ignite Team, “Openfire: Custom Database Integration Guide”, Ignite, http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/db-integration-guide.html
[22] iJab, http://www.ijab.im/
[23] Interactive Advertising Bureau Spain, http://www.iabspain.net/
[24] Internet Engineering Task Force (IETF), http://www.ietf.org/
[25] Internet Relay Chat, Wikipedia, http://es.wikipedia.org/wiki/Internet_Relay_Chat
[26] jQuery UI CSS Framework, http://jqueryui.com/docs/Theming/API
[27] jQuery UI, http://jqueryui.com/
[28] JQuery, http://jquery.com/
[29] JQueryUI ThemeRoller, http://jqueryui.com/themeroller/
[30] JSON, Wikipedia, http://es.wikipedia.org/wiki/JSON
[31] Live, http://www.live.com
[32] Mahara: Open source eportfolios, http://mahara.org/
[33] Marc Pransky, http://en.wikipedia.org/wiki/Marc_Prensky
[34] Microsoft Windows Live Messenger, http://www.microsoft.com/spain/windowslive/messenger.aspx
[35] Moodle, http://www.moodle.org

[36] Native digital, Wikipedia, http://en.wikipedia.org/wiki/Digital_native
[37] Nativo digital, Wikipedia, http://es.wikipedia.org/wiki/Nativo_digital
[38] Openfire Server, Ignite, http://www.igniterealtime.org/projects/openfire/
[39] Patrón Singleton, Wikipedia, http://es.wikipedia.org/wiki/Singleton
[40] Permisos de accesos a archivos, Wikipedia, http://es.wikipedia.org/wiki/Permisos_de_acceso_a_archivos
[41] RAID 1, Wikipedia, http://es.wikipedia.org/wiki/RAID#RAID_1
[42] RFC3920 - Extensible Messaging and Presence Protocol (XMMP): Core, IETF, http://tools.ietf.org/html/rfc3920
[43] RFC3920 - Extensible Messaging and Presence Protocol (XMPP), IETF, http://tools.ietf.org/html/rfc3920
[44] RFC3921 - Extensible Messaging and Presence Protocol (XMMP): Instant Messaging and Presence, IETF, http://tools.ietf.org/html/rfc3921
[45] Session Initiation Protocol (SIP), Wikipedia, http://es.wikipedia.org/wiki/Session_Initiation_Protocol
[46] Skype, http://www.skype.com
[47] Stan Schroeder, "Facebook Chat now works for everyone", Mashable, http://mashable.com/2008/04/23/facebook-chat-works/
[48] Tuenti Corporate, “Tuenti: Una conversación en tiempo real”, Tuenti Blog, http://blog.tuenti.com/tuenti-una-conversacion-en-tiempo-real/
[49] Tutor, http://tutor.ugr.es
[50] Twitter, http://www.twitter.com
[51] Unique URLs, AJAX Patterns, http://ajaxpatterns.org/Unique_URLs
[52] Video del sistema, Tutor, https://tutor2.ugr.es/files/video1.avi
[53] Web 2.0, Wikipedia, http://es.wikipedia.org/wiki/Web_2.0
[54] Webuzz, http://webuzz.im
[55] XMPP Extensions, XMPP Standards Foundation, http://xmpp.org/xmpp-protocols/xmpp-extensions/
[56] XMPP Servers, XMPP Standards Foundation, http://xmpp.org/xmpp-software/servers/
[57] XMPP4GWT: XMPP Library for the web, Tigase, http://www.tigase.org/project/xmpp4gwt
[58] Yahoo UI Library, Yahoo, http://developer.yahoo.com/yui/
[59] Yahoo, http://www.yahoo.com
[60] Youtube, http://www.youtube.com
[61] Zhan, "iJab parameters description", Ijab, http://opensource.ijab.im/node/89