ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERO EN INFORMÁTICA

DESARROLLO DE UN SISTEMA DE TEST DE CÁLCULO EN SIETTE

Realizado por

ALBERTO MANCHEÑO VILLENA

Dirigido por

RICARDO CONEJO MUÑOZ

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN (LCC)

UNIVERSIDAD DE MÁLAGA

MÁLAGA, 23 de Febrero 2007

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Ingeniería Superior en Informática

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente

Dº/Dª._____

Secretario

Dº/Dª._____

Vocal

Dº/Dª._____

para juzgar el proyecto Fin de Carrera titulado:

_____

_____

del alumno

Dº/Dª._____

dirigido por

Dº/Dª._____

ACORDÓ POR_____OTORGAR LA CALIFICACIÓN DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a de del 200_

El Presidente El Secretario El Vocal

Fdo: Fdo: Fdo:

A mi familia por todo su apoyo,
A mis amigos por las palabras de ánimo,
A Juan y a Eduardo por la inestimable ayuda,
A Ricardo, mi director de proyecto, por darme la oportunidad de finalizar mi carrera dentro de un gran proyecto

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1
Introducción

1 Introducción

En esta memoria del proyecto de fin de carrera "Desarrollo de un sistema de test de cálculo en SIETTE" se hace constar todo el proceso el desarrollo del mismo, explicando desde el origen del problema a la solución creada. Está dividida en siete Capítulos más los Anexos, que podemos agrupar en cinco partes bien diferenciadas: Introducción, Planteamiento, Implementación, Conclusiones y Bibliografía.

La primera, Introducción, comprende sólo este capítulo, donde se describe la estructura de la memoria y se avanza el contenido de los diferentes capítulos.

La segunda, el Planteamiento, se corresponde al Capítulo dos. En éste se explican los objetivos del proyecto, el planteamiento del problema inicial que dio lugar al proyecto, los antecedentes, el estado del problema antes de la realización del proyecto y definiciones de términos que aparecerán.

La Implementación, que incluye los capítulos tres, cuatro y cinco, así como los anexos, es la tercera de estas partes El Capítulo tres describe brevemente todas las tecnologías usadas en la implementación. El cuarto, el más extenso, explica todo el proceso software seguido para la implementación del proyecto. Seguidamente, el quinto capítulo, resume los contenidos de la batería tests creados en SIETTE [18] para la demostración del buen funcionamiento de este sistema, describiendo un Caso de Uso típico. Finalmente, los anexos, detallan información que, aún siendo útil, por su extensión no se incluyó en el cuarto capítulo, dado que dificultaría la lectura del mismo sobremanera, así como un manual de usuario de las aplicaciones implementadas.

Las conclusiones, el capítulo sexto, exponen los logros obtenidos, los problemas hallados así como las posibles mejoras.

Y para concluir, se incluye toda la bibliografía referenciada.

Capítulo 2
Motivación y Entorno

2 Motivación y Entorno

2.1 Planteamiento del Problema

Esta sección se divide en 4 partes. Primero una introducción al entorno en el que se inscribe el sistema Siette, seguido de una explicación general de las características de éste. A continuación detallamos qué es Wiris [16], una de las partes esenciales del proyecto, terminando la sección con un planteamiento completo del problema que se plantea.

2.1.1 Introducción

Internet ha supuesto desde su nacimiento una revolución que ha afectado a una infinidad de áreas del comportamiento humano. La educación y la docencia no son una excepción. Internet ha abierto las puertas a cantidades enormes de conocimiento, haciéndolos públicos y disponibles para quién quiera consultarlos. El acercamiento de estos conocimientos a través de las redes de la información a los usuarios finales es lo que ha propiciado esta revolución en los métodos de aprendizaje.

Pero la revolución que mencionamos no ha parado en éste punto. No basta sólo con acercar los conocimientos al usuario, o ampliarlos o mejorar los métodos de búsqueda. La docencia, aparte de la adquisición de éstos, consta de muchos otros aspectos como la relación entre el alumno y el profesor, el seguimiento de éste último para que el primero consiga sus objetivos, la evaluación del rendimiento del aprendizaje, y la disponibilidad de ayuda y consejo cuando sea necesario. Esto no se consigue sólo a través de la amplia disponibilidad de conocimientos. Hay que mimetizar el proceso de aprendizaje que se lleva a cabo en un aula, con un profesor y un alumno, y trasladarlo a Internet, usando los recursos y herramientas que estas nuevas tecnologías brindan. Esto es el e-Learning.

El e-Learning, o el "uso de nuevas tecnologías multimedia y de Internet para mejorar la calidad del aprendizaje mediante el acceso a recursos y servicios, y a colaboraciones e intercambios a larga distancia" [31] es una disciplina relativamente nueva. Iniciado por las empresas y sus departamentos de "training" para mejorar la formación y el rendimiento de los trabajadores, al principio sólo contaba con soportes

multimedia (texto, video y sonido en soportes magnéticos u ópticos) que se creaban exclusivamente para un proceso de formación específico, y se usaban en los propios centros, con personal y máquinas dedicadas a ello. Más adelante, con la llegada de Internet, se pasó a la creación de contenidos en servidores especializados, accesibles desde cualquier punto de la red. Los inconvenientes iniciales de lentitud en los accesos o carencia de herramientas adecuadas han ido desapareciendo paulatinamente. Hoy en día, se utiliza una mezcla de aprendizaje a distancia a través de Internet con clases presenciales. A esto ha contribuido el nutrido grupo de empresas que vieron las posibilidades de éste mercado (ofrecer e-learning a otras empresas) [32].

Paralelamente, en las Universidades, también se desarrollaron sistemas de e-Learning [33], basados casi todos en sistemas de código abierto (open source). Citar por ejemplo, Moodle [37], una plataforma open source para el desarrollo de contenidos basados en un aprendizaje en el que el alumno toma gran parte de la iniciativa.

En ambos casos, el desarrollo de contenidos está siempre enfocado a la plataforma para la que se quiere desarrollar. Existen algunos esfuerzos para mejorar la interoperabilidad, como SCORM [35] o QTI [34], que permiten el intercambio de contenidos entre diferentes sistemas, y soportados por una amplia variedad de herramientas de autor.

Podemos decir que una herramienta de autor es una herramienta software que los desarrolladores usan para crear contenidos para que sean usados por usuarios finales. Aunque estas herramientas tienen un amplio rango de usos, son normalmente usadas para crear módulos de e-learning. Un editor web, Microsoft PowerPoint o Flash podrían entrar en la categoría de herramientas de autor, pero la definición más estricta sólo concierne a aquellas herramientas que desarrollan contenidos de e-learning que siguen algún estándar, como los antes nombrados.

2.1.2 SIETTE

Siette es un módulo de diagnóstico, es decir, una herramienta de enseñanza que aplica técnicas de Inteligencia Artificial para diagnosticar el conocimiento del estudiante durante el proceso de instrucción. Es, asimismo, un entorno de aplicaciones web para la construcción y generación de tests. Principalmente, permite construir y

administrar Tests Adaptativos Informatizados (TAI), aunque es un sistema que también ofrece la posibilidad de administrar tests convencionales basados en la Teoría Clásica de Tests (TCT). A través de una interfaz web, los alumnos pueden realizar tests para autoevaluarse; o bien los profesores pueden evaluar a sus alumnos, utilizando SIETTE como medio de evaluación académica [1].

Figura 2.1: Página de entrada a Siette

Para construir y modificar los tests así como su contenido, SIETTE ofrece a los profesores un entorno de herramientas de autor. Este entorno permite además analizar las sesiones de tests llevadas a cabo por los alumnos.

Este sistema puede funcionar bien como una herramienta de evaluación independiente, o integrado dentro de otros sistemas web educativos, especialmente en sistemas de enseñanza adaptativos, en los que SIETTE puede desempeñar el rol de sistema de diagnostico del conocimiento del alumno. Es además multilingüe (actualmente disponible en español, ingles y alemán) y abierto a la inclusión de otros idiomas, gracias a la generación dinámica de sus interfaces.

La primera versión del sistema SIETTE fue construida en torno a 1998 como resultado del proyecto de fin de carrera realizado por Antonia Ríos y dirigido de forma conjunta por los doctores José Luís Pérez de la Cruz y Ricardo Conejo. Cada test permitía únicamente evaluar en solo concepto. La interfaz del aula virtual se ha

mantenido prácticamente igual hasta nuestros días. Este sistema fue implementado en lenguaje C, utilizando la tecnología de CGI combinada con PHP [36], un lenguaje de programación embebido en HTML [20] para la generación dinámica de páginas web. La base de conocimiento de aquel sistema fue implementada sobre un gestor de base de datos relacionales PostgreSQL [38].

Posteriormente, se añadieron nuevas funcionalidades. El núcleo actual del sistema SIETTE fue implementado partiendo desde cero, siguiendo un nuevo diseño, una nueva metodología de programación, sobre un lenguaje de programación y un sistema gestor de base de datos completamente diferentes. Este nuevo núcleo comenzó a desarrollarse en la segunda mitad del año 2000. A partir de ese momento se han ido introduciendo mejoras y corrigiendo errores, generándose en consecuencia subsecuentes versiones del sistema, como consecuencia de la evaluación formativa que ha sufrido el sistema, y que se describiría en el capítulo siguiente. En la actualidad, aunque se dispone de una versión estable del sistema, se siguen realizando nuevas actualizaciones y mejoras. SIETTE se utiliza con cierta asiduidad en la administración de tests de evaluación en asignaturas de diversas titulaciones de carácter universitario.

SIETTE se basa en la creación de contenidos como asignaturas. Cada asignatura está compuesta de temas, que a su vez pueden contener a otros temas. Los temas a su vez contienen ítems, que son unidades mínimas de conocimiento y evaluación. Asimismo, cada asignatura puede tener varios tests que evalúan un conjunto parcial o total de los ítems definidos en la misma. Todos los ítems, junto a los temas, los tests y las asignaturas a las que pertenecen, constituyen la base del conocimiento, es decir, los contenidos.

Cada alumno puede acceder al aula virtual, o aula de tests y realizar los que hay disponibles sobre los contenidos de la base del conocimiento. SIETTE guarda un registro de estas evaluaciones para poder analizar los resultados y presentarlos al profesor que creó los contenidos.

Además, guarda la información relativa a los profesores, y su relación con los contenidos creados y sus alumnos.

Su arquitectura se muestra en el siguiente diagrama:

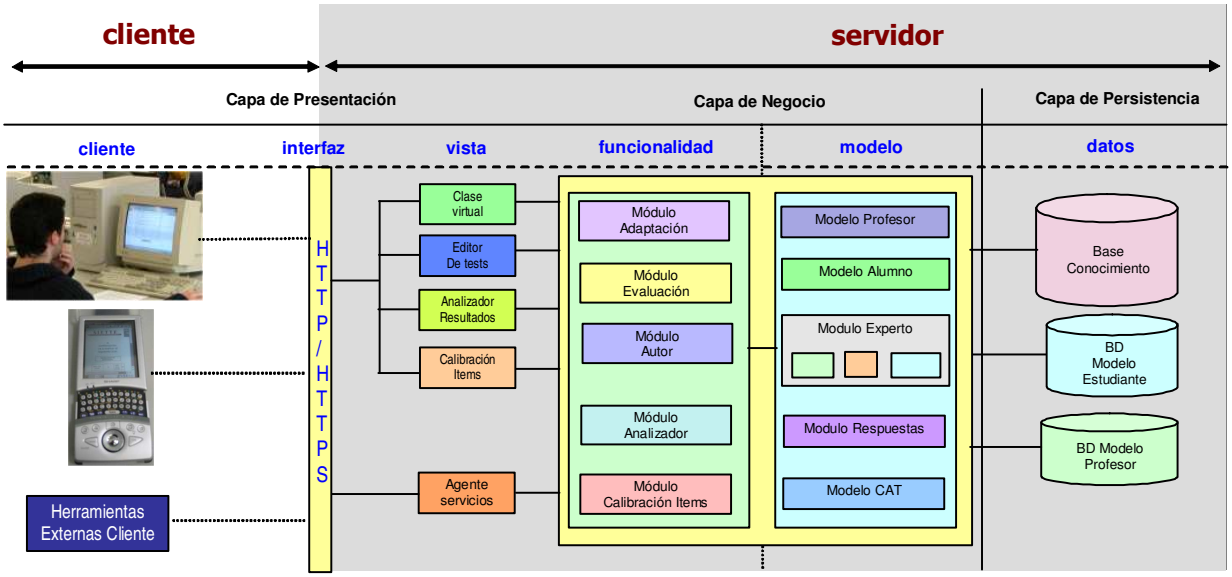


Figura 2.2: Arquitectura de SIETTE

Siette es una aplicación web estructurada en capas. La primera capa y más interna, la capa de persistencia, la que ya hemos comentado, que es la que almacena los datos. La segunda, o capa de negocio, implementa los diferentes módulos que luego se utilizan en la capa de presentación. Estos son:

- Módulo Evaluación: Se encarga de la elección de ítems de la base de conocimiento, en base a los criterios establecidos por el profesor, la corrección de la respuesta del alumno y su posterior almacenamiento.
- Módulo Adaptación: Se encarga de establecer los criterios de elección de ítems cuando se trata de un test adaptativo.
- Módulo Autor: Es el encargado de manejar todo el proceso de creación de contenidos a través de las herramientas de autor facilitadas en la capa de presentación.
- Módulo Analizador: Se encarga de procesar la información almacenada en la base de conocimiento y en el modelo del alumno y presentar los resultados de las evaluaciones.
- Módulo Calibrador de Ítems: Calibra las características de los ítems para reflejar su historial de sesiones de tests dentro del módulo de adaptación.

La tercera capa, o capa de presentación, es la que interacciona con el usuario final. De todos sus componentes, dos son los más importantes, el Aula Virtual y el Editor de Tests.

El Aula Virtual es una interfaz web donde el alumno puede acceder y realizar los diferentes tests que los profesores han creado de las diferentes asignaturas. El módulo de Evaluación (y el de Adaptación, si se da el caso) van eligiendo diferentes ítems a evaluar y se presentan al alumno para que este vaya respondiendo.

El Editor de Tests es una herramienta de autor que permite la edición y creación de contenidos, siendo también otra una interfaz web. Siette cuenta actualmente con dos de estas herramientas, TEDI y AthoS. Ambas sirven para el mismo propósito, siendo la última la más recientemente desarrollada.

Los otros dos componentes son el Calibrador y el Analizador. Mientras que el primero calibra los parámetros para la generación de tests adaptativos el segundo muestra por pantalla, también a través de una interfaz web todo los datos que produce el módulo Analizador, pudiendo el profesor navegar por ellos.

Tipos de ítems

Como se ha comentado, Siette permite a los profesores crear y editar contenidos en forma de asignaturas, temas e ítems. No todos los ítems son iguales ya que hay varios tipos que se diferencian en el tratamiento que les da Siette.

Todos los ítems cuentan con dos partes básicas, el enunciado y la respuesta. El enunciado es una parte común a todos ellos, y como su nombre indica, contiene el texto donde se plantea la pregunta o ejercicio al alumno. El profesor puede incluir en el mismo código HTML para formatear el texto o incluir elementos externos como applets (tanto de los applets como de HTML se hablará extensamente en el capítulo tercero de esta memoria).

La parte de la respuesta es la que diferencia a los tipos de ítems. En Siette podemos encontrar actualmente tres clases bien diferenciadas.

Ítems de Respuesta Única: son aquellos que incluyen varias posibilidades, de las cuales, sólo una es correcta, el resto son incorrectas. Al alumno, en el Aula de Tests, le aparecen varias posibilidades, de las cuales sólo puede marcar una de ellas. El profesor deberá definir al menos dos respuestas, una correcta y una incorrecta. Las preguntas de verdadero/falso son una subclase de este tipo de ítems.

Figura 2.3: Pregunta de Respuesta Única

Ítems de Respuesta Múltiple Independiente: son aquellos que incluyen varias posibilidades, de las cuales, varias pueden ser correctas, y varias incorrectas. Al alumno, en el Aula de Tests le aparecen varias posibilidades, de las cuales sólo podrá elegir un número igual al número de respuestas correctas. Cada respuesta correcta elegida puntuará por sí sola (por eso es Independiente, si fuesen Dependientes, lo que puntuaría sería la elección del conjunto de respuestas correcto), mientras que la elección de una respuesta incorrecta penalizará.

Figura 2.4: Pregunta de Respuesta Múltiple

Ítems de Respuesta Corta: son aquellos que permiten al alumno responder lo que el estime conveniente. En el Aula de Tests se le presenta el enunciado seguido de un pequeño hueco en blanco, donde a través del teclado, el alumno introduce su respuesta. La corrección de la misma corre a cargo de los Patrones de Corrección, que comparan la respuesta introducida con aquellos patrones que el profesor ha definido. Los patrones de corrección se comentan a continuación.

Figura 2.5: Pregunta de Respuesta Libre

Tipos de patrones

Para los ítems de respuesta corta existen varios tipos de patrones de corrección. Una respuesta es correcta si entra en ese patrón de corrección.

Todos los patrones tienen la opción de normalizar la respuesta del alumno a través de la activación de cuatro opciones:

- Modo Mayúsculas: Cuando esta activado, el sistema no distingue entre mayúsculas y minúsculas
- Modo Blancos: Cuando está activado, el patrón ignora los espacios en blanco tanto del patrón como de la respuesta
- Modo Acentos: Cuando está activado el patrón reconoce como validas las vocales acentuadas o no, lo estén o no en el patrón. Asimismo, para compatibilidad con otros teclados la letra Ñ se reconoce como N en caso de que esta opción este activada.
- Modo Signos de Puntuación: Se ignoran todos los signos de puntuación. Hay que tener cuidado si se usa este modo en combinación con secuencias de dígitos para representar números reales, ya que el punto decimal es considerado en este caso como un signo de puntuación, y es ignorado, lo que puede dar lugar a reconocimiento es incorrectos (por ejemplo, 333 puede reconocerse como 3.33).

Éstas modifican la respuesta del alumno antes de compararla con el patrón establecido por el profesor (que también es normalizado). Se pueden establecer tres tipos de patrones en SIETTE:

Patrón de Correspondencia: Es el patrón más sencillo de todos. Simplemente confronta la respuesta del alumno ya normalizada con la establecida por el profesor.

Patrón Siette: Están definidos mediante un lenguaje especifico, similar en algunos casos a los lenguajes que se usan para definir expresiones regulares en los analizadores léxicos, y que recuerda en gran medida a las convenciones utilizadas para los nombres de ficheros. Una patrón esta compuesto por una secuencia de caracteres. Los caracteres pueden ser de dos tipos:

Caracteres especiales: *** ( ) [ ] { } ? | \ $ #**

Caracteres ordinarios:Todos los demás caracteres ASCII del 32 en adelante.

En un patrón, un carácter ordinario representa al propio carácter. Para representar los caracteres especiales es necesario utilizar el carácter **** antepuesto. Por ejemplo

Patrón	Respuesta
A	A
*****	*
****	\

La concatenación de caracteres en el patrón indica concatenación de caracteres en la respuesta:

Patrón	Respuesta
Abc	abc
a*c	a*c
****	\\\

El carácter especial **|** representa la alternativa o disyunción, se usa para combinar posibles respuestas en el patrón, por ejemplo:

Patrón	Respuesta

a|e|i|o|u	U
Altura|altitud	altura
A|AB|ABA	AB

Pueden usarse paréntesis para agrupar expresiones y formar otras más complejas:

Patrón	Respuestas
alt(o|a)	Alto
(C|c)ol(o|ó)n	Colón
(pa|ma|ra)|(ta|la)	Pala

Los paréntesis angulares, sirven para denotar disyunciones de un solo símbolo, es decir, son simplemente modos abreviados de escribir expresiones que podrían escribirse con paréntesis:

Patrón	Respuesta
[aeiou]	U
Alt [ao]	alta
[Cc]ol [oó]n	colon

Las llaves **{}** tienen también un significado especial. Las expresiones que se sitúan entre llaves se entiende que son opcionales, es decir, que pueden aparecer o no en la respuesta:

Patrón	Respuesta
{Cristobal} Colón	Colón
{a}{b}{c}=1	ab=1
30{km|kilómetros|k}	30km

El carácter especial **?** representa a un símbolo cualquiera, sin especificar cual (similar al significado que tiene para nombres de ficheros):

Patrón	Respuesta
X=?	X=1
Alt?	alt;
A??B	A+-B

El carácter especial ***** representa a una secuencia de símbolos cualquiera, de longitud cero o mayor que cero (similar al significado que tiene para nombres de ficheros)

Patrón	Respuesta
***Colón**	Cristobal Colón
m*	Metros
A*B	A1233456767B

Y por último, el carácter especial **$** indica el final de la expresión correspondiente al patrón. Su usa internamente, por lo que debe utilizarse la secuencia **\$** para incluir este carácter en las respuestas.

Expresiones Regulares de JAVA: Este tipo de patrón corresponde a la especificación de expresiones regulares que se implementaron en el lenguaje de programación JAVA a partir de su versión 1.4. Esta especificación, que se puede encontrar en [39], es muy completa y extensa. A continuación mostramos las características más generales de la misma:

Caracteres válidos principales que acepta el patrón

x	The character x
\\	El character '\'
\t	El carácter de tabulación (UNICODE: '\u0009')
\n	El carácter nueva línea (UNICODE: '\u000A')

\r	El carácter de retorno de carro (UNICODE: '\u000D')

Clases de caracteres principales:

[abc]	a, b, ó c (clase simple)
[^abc]	Cualquier carácter excepto a, b, o c (negación)
[a-zA-Z]	De a hasta z o desde A hasta Z, inclusive (rango)
[a-d [m-p]]	De a hasta d, o desde m hasta p: [a-dm-p] (unión)
[a-z&& [def]]	d, e, o f (intersección)
[a-z&& [^bc]]	De a hasta z, excepto b y c: [ad-z] (substracción)
[a-z&& [^m-p]]	Desde a hasta z, y no desde m hasta p: [a-lq-z](substracción)

Además incluye la posibilidad de incluir bloques ya predefinidos como los números o los alfanuméricos. Acepta un gran número de operadores, de los cuales, los más importantes son:

Cuantificadores:

X?	X, una vez o ninguna
X*	X, ninguna o más veces
X+	X, una o más veces
X{n}	X, exactamente n veces
X{n,}	X, al menos n veces
X{n,m}	X, al menos n veces pero no más de m

Y los siguientes:

XY	X seguido de Y
X|Y	Ó X ó Y

Cualquier expresión puede ser rodeada por paréntesis y cualquier operador afectará a todo el conjunto.

También existe la posibilidad de no establecer ningún patrón.

Características comunes de los ítems

Hay algunas características que hacen que el Aula de Tests presente de forma diferente los ítems que hemos definido.

Ítem Externo: Aquel ítem que su presentación está guardada en una página HTML externa a Siette.

Ítem Auto-respondido: Aquel ítem cuya corrección no se recoge a través del código HTML presentado en el Aula de Tests, sino que incluyen su propio mecanismo, como por ejemplo la inclusión de applets que recogerán la respuesta del alumno (en vez de marcarla o escribirla) y la devolverán a los mecanismos de evaluación del Aula Virtual.

Ítem Activo: Aquel ítem que puede entrar a formar parte de un test que se realice en el Aula de Tests.

Por lo tanto, esta es toda la variedad de opciones que ofrece Siette para crear contenido. La gran parte de las veces, los contenidos se escriben en texto plano, o incluyéndolo dentro de código HTML, lo cual restringe la capacidad expresiva de los enunciados a lo que esta tecnología permite.

2.1.3 Wiris

Wiris [19] es un software que aúna un conjunto de aplicaciones relacionadas con la representación de contenido matemático en interfaces de usuario. Desarrollado en el lenguaje de programación Java [11] por la empresa MathForMore, incluye una amplia variedad de funcionalidades para crear, mostrar y editar gráficamente contenido matemático a través de interfaces de usuarios, incluir estas dentro de otros programas desarrollados en Java o dentro de código HTML. Utiliza el estándar emergente para la representación de contenido matemático OpenMath [5] (explicado en el capítulo 3), aunque internamente utilice MathML [20] para su funcionamiento (igualmente detallado en el capítulo 3).

2.1.4 Planteamiento del problema

Por lo tanto, hemos visto que Internet ha supuesto una revolución para el aprendizaje, y que Siette ha sido partícipe de ello. Como se ha comentado, Siette al estar

basado en la presentación de sus contenidos a través de páginas webs, restringe su expresividad a la hora de crear estos contenidos a la del lenguaje HTML.

Desde siempre, una de las áreas donde más se ha justificado la interacción alumno profesor son las Matemáticas. Dado su amplio espectro de áreas, y su especialización al usar su propia terminología, sus propios caracteres y representaciones de la información, es un área difícil de tratar a través del e-learning.

También la visualización de esta terminología, caracteres y representaciones a través de páginas web hacen que sea muy dificultoso en Siette crear contenido relacionado con las matemáticas.

Lo que se persigue en este proyecto es, por tanto, ampliar la funcionalidad de Siette para poder crear en él contenido matemático (en este caso en el área de Cálculo), utilizando para ello el software de Wiris.

2.2 Definiciones

A lo largo de este proyecto nombraremos varias veces los siguientes conceptos:

Igualdad sintáctica: La igualdad sintáctica indica la relación entre dos fórmulas matemáticas las cuales son idénticas elemento a elemento, teniendo en cuenta el orden de la aparición de estos. Es decir $x+2$ es igual sintácticamente sólo a $x+2$, mientras que no es igual sintácticamente a $x+3-1$, a $2+x$ ó a $\frac{x^2}{x}+2$.

Igualdad semántica: La igualdad semántica indica la relación entre dos fórmulas las cuales pueden sustituirse mutuamente sin alterar el significado o resolución del todo al que pertenecen. Es decir $x+2$ es igual semánticamente a $2+x$, a $\frac{x^2}{x}+2$ o a $x+3-1$.

Sistema Tutor Inteligente: Un STI es un SEAO (Sistema de Enseñanza Asistida por Ordenador) que utiliza técnicas de IA, principalmente para representar el

conocimiento, y dirigir una estrategia de enseñanza; y que es capaz de comportarse como un experto, tanto en el dominio de conocimiento que enseña (mostrando al alumno como aplicar dicho conocimiento), como en el dominio pedagógico, donde es capaz de diagnosticar la situación en la que se encuentra el estudiante, y de acuerdo a ello, ofrecer una acción o solución que le permita progresar en el aprendizaje.

Test Adaptativo Informatizado (TAI): Los Tests Adaptativos Informatizados (TAI), tienen tras de sí un sólido fundamento teórico que garantiza la validez, viabilidad y objetividad de las evaluaciones que realizan. A través de un TAI se obtiene como resultado la estimación del nivel de conocimiento del alumno. Requieren un menor número de preguntas por test con respecto a los tests convencionales. Esto es debido a que cada pregunta es seleccionada en función de la estimación que hasta ese momento ha realizado el test del conocimiento del alumno. Los TAI se basan en aplicar un algoritmo de evaluación, el cual se sustenta principalmente en una teoría psicométrica denominada Teoría de Respuesta al Ítem (TRI). Según ésta, la respuesta que un examinando da a una pregunta, está relacionada con el nivel de conocimiento que éste posee. Esta relación se cuantifica mediante una o más funciones de densidad denominadas curvas características.

Teoría Clásica de Tests (TCT): Se fundamenta en la suposición de que la puntuación observada de un alumno se compone del valor real obtenido más la medida del error, las cuales no están correlacionadas.

CAS (Computer Algebra System): Un Sistema Algebraico Computarizado es un programa de ordenador que facilita el cálculo matemático simbólico, pues permite la manipulación de expresiones matemáticas en forma simbólica.

2.3 Objetivos

Desarrollando el planteamiento del problema que se quiere resolver en este proyecto, que se ha enunciado en la sección 2.1.4, podemos elaborar la siguiente lista:

Habrá que crear un editor que permita crear y editar enunciados de preguntas que contengan fórmulas matemáticas y se integre con los ya existentes editores externos

que posee Siette y que permita formatear el texto añadiéndole código HTML. Este editor se integrará dentro de Siette de la misma manera que el resto de editores externos, compartiendo sus comportamientos comunes de comunicación con las dos herramientas de autor existentes, AthoS y TEDI. Para la representación de las fórmulas matemáticas se utilizará el software Wiris.

Igualmente habrá que crear un editor de las respuestas, que permita al profesor crear o editar una fórmula matemática que sea solución del enunciado propuesto. Al igual que el editor anterior, se integrará dentro de Siette con el resto de editores externos, manteniendo el comportamiento y comunicación comunes. Esta fórmula estará representada en el estándar OpenMath.

Se tendrá que desarrollar un sistema que permita al alumno escribir fórmula matemática como solución, utilizando el estándar OpenMath, y que permita posteriormente la visualización de la misma dentro del contexto del Aula de Tests, utilizando los canales de comunicación que establece Siette para ello.

Una vez que el alumno ha respondido, habrá que evaluar su respuesta y compararla con la que el profesor ha establecido. Para ello se usará la igualdad semántica. Es por eso que habrá que crear un nuevo tipo de patrón de corrección asigne esta igualdad. Con este propósito se creará un evaluador numérico que compare fórmulas en el estándar OpenMath.

Finalmente, y para probar la idoneidad del sistema desarrollado para la educación y el desarrollo de test de índole matemática, así como para mostrar todas las funcionalidades implementadas, se creará en el sitio público de Siette, una nueva asignatura que contenga ítems de cálculo matemático (sobre límites, integrales, series, derivadas, etc. …).

2.4 Antecedentes

En la actualidad existen una gran cantidad de sistemas que permiten la realización de tests tanto convencionales (basados en la TCT, porcentaje de respuestas o suma de puntos) como basados en la TAI. Intralearn [48], WebAssessor [49] o WebCT

[50] son los más destacados del primer grupo, mientras que Terranova CAT [51] o CAT Global [52] lo son del segundo grupo.

En el plano de los Sistemas Tutores Inteligentes podemos hacer una división en varios grupos:

- STI con modelos de evaluación basado en heurísticos, como LeActiveMath [22], Quanta [53] o ALEKS [21].
- STI con modelos de evaluación basados en tests adaptativos, como INSPIRE [54], SKATE [55] o PASS [56].
- STI con modelos de evaluación basados en lógica difusa, como ALICE [57] e INSPIRE [54]
- STI con modelos de evaluación basados en redes bayesianas, como ANDES [58] y POLAS [59].

De todos estos sistemas, hemos querido centrarnos en dos, que tienen un amplio potencial para la creación y evaluación de contenido matemático de forma muy visual, que es precisamente lo que se busca en este proyecto. Los dos elegidos son ALEKS y LeActiveMath. De estos dos, describiremos de manera general sus características y nos centraremos en las posibilidades que ofrecen con respecto a contenido matemático.

2.4.1 Aleks

ALEKS es un sistema de aprendizaje y evaluación basado en la web y en técnicas de inteligencia artificial. ALEKS usa evaluación adaptativa para determinar rápidamente y de forma precisa lo que un estudiante conoce y desconoce de un curso. Luego, ALEKS instruye al estudiante en los temas que más rápido va a prender. A lo largo del curso, ALEKS periódicamente re-evalúa al estudiante para asegurarse que los conocimientos han sido realmente adquiridos.

Figura 2.6: Logotipo de ALEKS

Se originó en investigación en la Universidad de Nueva York y la Universidad de California, Irving, por un equipo de ingenieros software, matemáticos y científicos del conocimiento con el apoyo de una beca multimillonaria de la Fundación Nacional de Ciencia.

Está basado en un trabajo originalmente teórico en un campo de estudio llamado "Teoría de Conocimiento del Espacio". Comenzado al principio de la década de los 80 por el Dr Jean-Claude Falmagne, un matemático de renombre mundial y Catedrático de Ciencias Cognitivas, que es actualmente Presidente y Fundado de la corporación ALEKS.

ALEKS evita las preguntas de respuesta múltiple, utilizando respuestas abiertas muy fáciles de responder a través de un gran conjunto de herramientas que imitan a lo que se haría con lápiz papel.

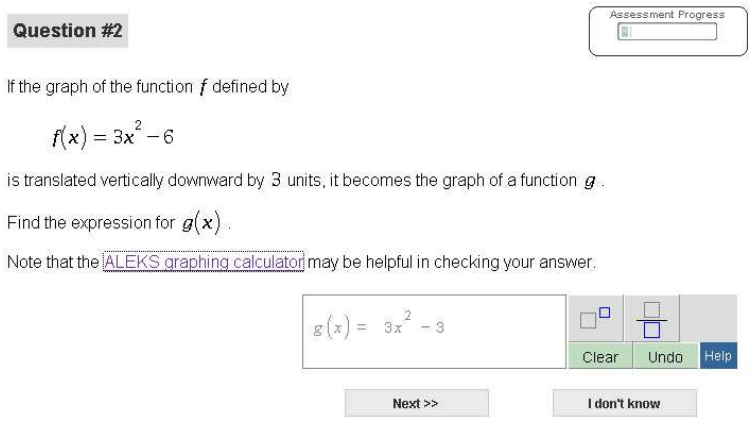


Figura 2.7: Pregunta de Respuesta Libre

Están por ejemplo las entradas para responder a preguntas sobre funciones, con gran posibilidad de componer fórmulas:

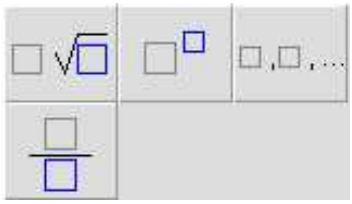


Figura 2.8 Detalle de los controles de entrada

Además de estar acompañado de una calculadora con pantalla gráfica que sirve de ayuda al estudiante.

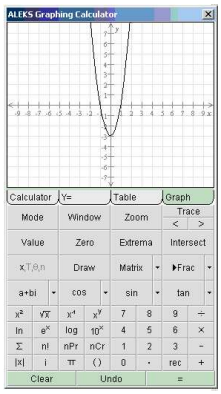


Figura 2.9 Calculadora gráfica de apoyo al estudiante

Algunos ítems incorporan la posibilidad de dibujar la función que se está pidiendo, o de responder con un valor de un ángulo dibujándolo en un círculo.

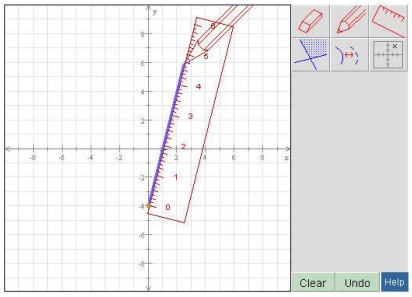


Figura 2.10: Interfaz para el trazado de funciones

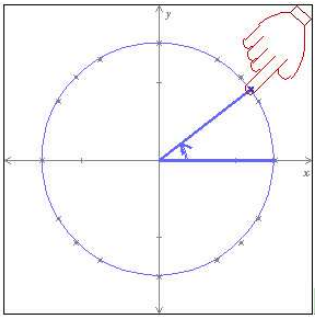


Figura 2.11: Interfaz para insertar valores de un ángulo

Al estudiante inicialmente se le presentan una serie de preguntas, normalmente unas 20 ó 30 para estimar su nivel inicial de conocimientos. Después de esto, el sistema genera un diagrama de sectores con los conocimientos estimados, y permite la práctica de los temas y contenidos donde se necesita refuerzos a través de las mismas interfaces gráficas que se presentaron en el test.

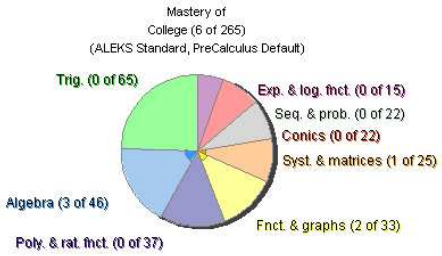


Figura 2.12: Diagrama de conocimientos del estudiante

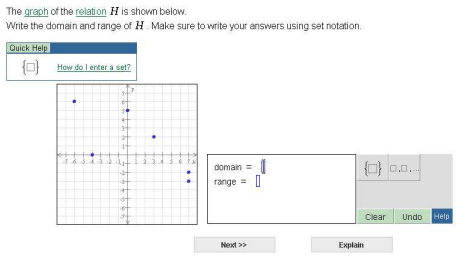


Figura 2.13 Módulo de práctica y aprendizaje

ALEKS está completamente basado en Java, y necesita de la instalación de un pequeño programa cliente. Luego, a través de la Web se puede acceder en cualquier momento y lugar al sistema.

Desafortunadamente, es un sistema comercial, del que sólo se ha podido probar una pequeña versión a modo de presentación, de modo que se intuye la posible edición de contenidos y la facilidad para la misma a través de las preguntas respondidas.

2.4.2 LeActiveMath

LeActiveMath es un entorno de aprendizaje adaptativo al usuario y basado en la web para las matemáticas. Genera dinámicamente cursos de material de estudio para aprender individualmente de acuerdo con los hitos y metas marcados, las preferencias y el dominio de los conceptos. Además hace sugerencias en cómo proceder con el aprendizaje y la navegación [8].

Desarrollado bajo el proyecto LeActiveMath, bajo el 6º Framework Program de la Comunidad Europea, a través de la colaboración de varias Universidades Europeas.

Su arquitectura, brevemente explicada, se basa en un modelo de un modelo del estudiante, un conjunto de reglas y técnicas de IA para el modelo pedagógico, y los contenidos, creados de forma modularizable. Los flujos de información de contenido

matemático están basados también en el estándar OpenMath, que se evalúan a través de un servicio externo de CAS [6].

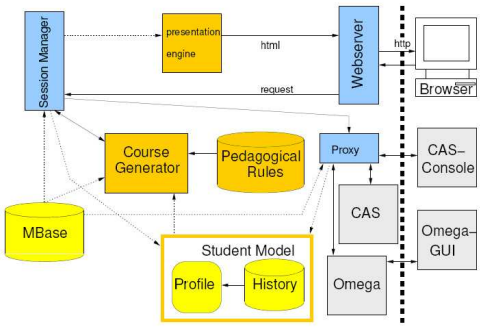


Figura 2.14: Arquitectura de LeActiveMath

Al iniciar una sesión en LeActiveMath, se puede elegir entre crear un libro para un aprendizaje específico o elegir alguno ya creado y público. Al alumno se le muestra el temario completo a la izquierda de la interfaz, a modo de índice, y a la derecha, la página del "libro" en el que se encuentra.

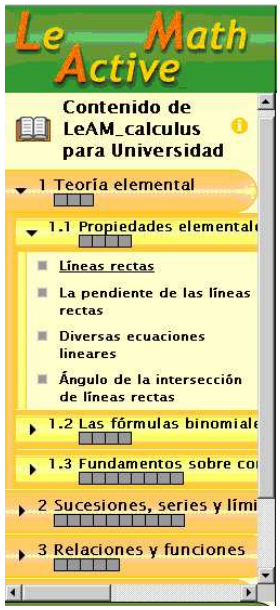


Figura 2.15: Índice del libro LeAM_Calculus

LeActiveMath incluye varios elementos de aprendizaje como Teoremas, Demostraciones, Conceptos, Ejemplos, Ejercicios… etc. El alumno puede elegir cualquier parte del libro para empezar.

La realización de un ejercicio se realiza en una ventana aparte, que incluye un editor de fórmulas muy parecido al que se usará en este proyecto (es el mismo

software). El sistema te corrige y te muestra nuevos ejercicios conforme se van completando las fases.

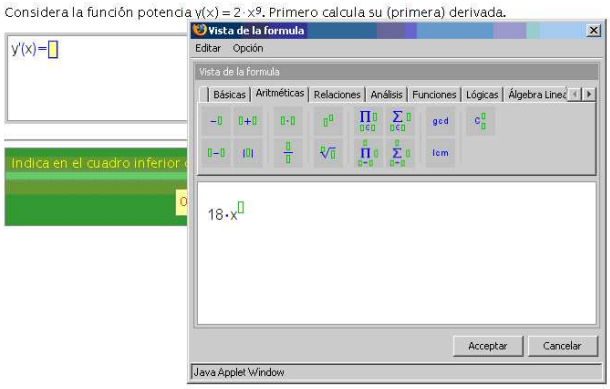


Figura 2.16: Ejercicio en LeActiveMath con Editor de Entrada

Una vez terminado, el sistema determina por lo respondido qué áreas hay que desarrollar más y en cuales se tiene todavía que mejorar a través de un código de colores.

Incluye además herramientas externas, como un trazador de funciones, un simplificador de expresiones o un cuaderno de anotaciones.

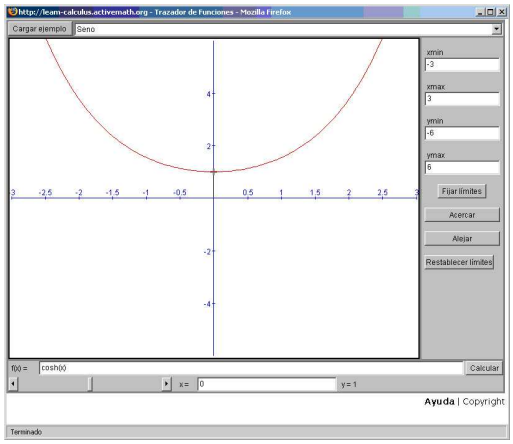


Figura 2.17: Trazador de funciones de ActiveMath

Podemos resumir, por lo tanto, que la interfaz web está basada completamente en HTML, el resultado de una traducción de los contenidos (escritos en OMDOC, basado en XML) y en el uso de Applets de Java. No permite la edición de contenidos, ya que esto es tarea de los desarrolladores del proyecto. El tratamiento del contenido matemático está perfectamente insertado en la interfaz y es de fácil uso para el alumno.

2.5 Estado previo del problema

Una vez analizados los objetivos del proyecto, habiendo observado el entorno del mismo y comparado con otras dos solucionas ya existentes a un problema parecido, pasamos a dar una visión más general del entorno de trabajo a partir del cual se va a empezar a trabajar.

Hemos hablado en el planteamiento del problema de los dos grandes componentes del proyecto, SIETTE, y Wiris, desde un punto de vista general.

En esta sección tomaremos el punto de vista de un proyecto software que pretende aunar estos dos componentes para lograr una solución al problema planteado en los objetivos.

2.5.1 Siette

Al inicio del proyecto, SIETTE estaba en un proceso de ampliación y desarrollo. Se estaban ultimando la implementación del nuevo Editor de Tests (AthoS), creando la herramienta de importación y exportación al estándar QTI comentado en la sección 2.1, y sufriendo un proceso general de mejora del rendimiento.

Una de las tareas dentro de este proyecto de fin de carrera es integrar la solución creada dentro de los procesos software de este desarrollo de SIETTE. De los detalles de esta integración se hablará en la sección 4.4. Ahora pasaremos a comentar las características software previas al comienzo del proyecto.

Como se comentó en la sección 1.1, SIETTE dispone de tres tipos de diferentes ítems. De estos, el de Respuesta Corta o Respuesta Libre es el que nos interesa, pues entre los objetivos de este proyecto está un nuevo Patrón de Corrección que compare semánticamente dos fórmulas matemáticas representadas en el estándar OpenMath.

En este tipo de ítems el patrón se introduce como un texto de respuesta y el tipo de patrón se elige de entre las opciones existentes.

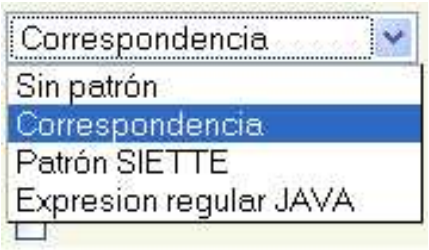


Figura 2.18: Tipos de Patrones

El patrón se puede definir como correcto o incorrecto. SIETTE permite editar cualquier texto a través de llamadas a Editores externos. Esto es aplicable a los textos de las respuestas, los patrones o los enunciados.

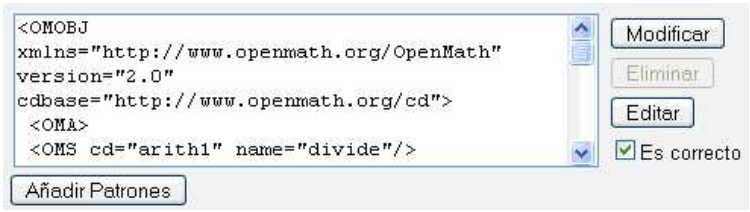


Figura 2.19: Recuadro de edición del patrón

Al comienzo del proyecto existían cuatro Editores Externos. Todos los editores externos son Applets de Java (véase siguiente capítulo) que cumplen unas interfaces específicas de comunicación con SIETTE. Dos de ellos eran de propósito general, y los otros dos específicos de un contexto, como se verá. Éstos editores eran:

Hermes: Es un editor de textos de propósito general que formatea el texto para presentarlo visualmente como HTML. Permite las opciones básicas de edición.

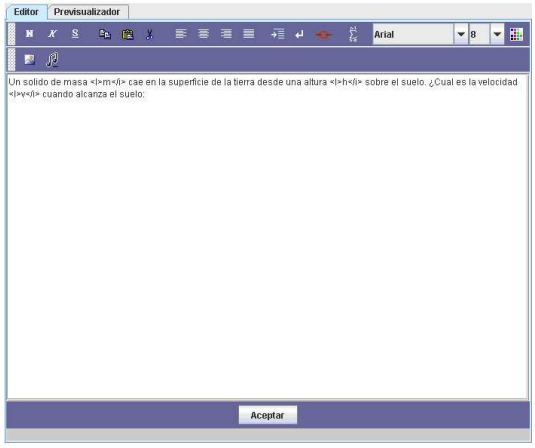


Figura 2.20: Editor Hermes

Edie: Es un editor de propósito general que compone imágenes a través de la conjunción de varios elementos, como fondos, leyendas, títulos, etc.

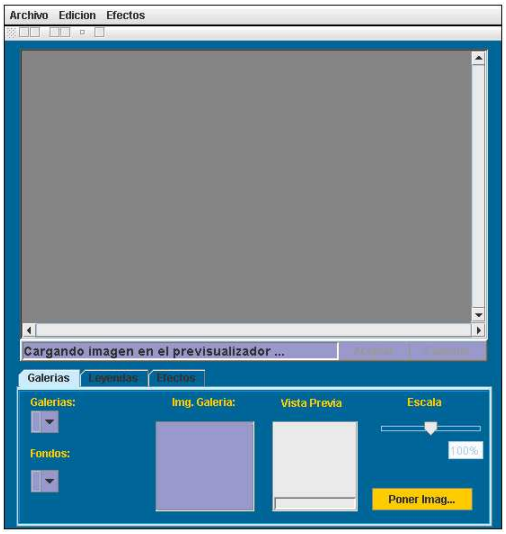


Figura 2.21: Editor Edie

SQLSiette: Es un editor específico para ítems de asignaturas de Base de Datos. Permite crear enunciados que sean la resolución de una consulta en el lenguaje SQL.

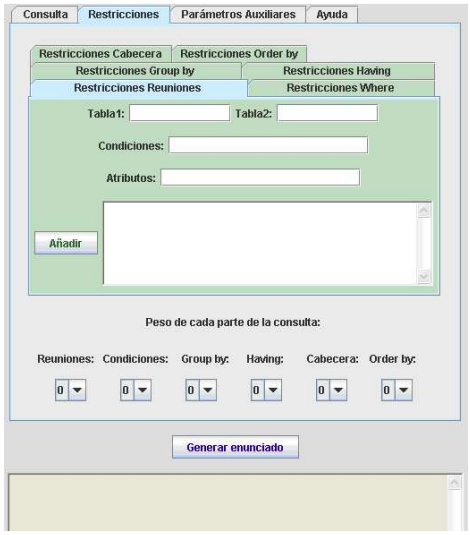


Figura 2.22: Editor SQLSiette

PLSiette: Es otro editor de propósito específico que se utiliza en los enunciados de los ítems de la asignatura de Procesadores de Lenguajes.

El editor a usar es un parámetro global de la asignatura. Son mostrados fuera del área del Editor de Tests, en una ventana flotante. Los editores que se han de crear para este proyecto han de emular la integración de estos editores.

Figura 2.23: Opciones de Editor

Por último, otro aspecto concerniente a este proyecto es la secuencia de pasos que un alumno ha de dar para responder un ítem Auto-Respondido, es decir, que se basa en el uso de applets para generar la respuesta. Normalmente, en el Aula de Test, SIETTE accede a los campos de formulario que el alumno ha respondido o rellenado y los envía al motor de evaluación. Pero al tratarse de applets los que generan las respuestas, la comunicación entre el motor de evaluación y el applet han de estar predefinidas y han de ser siempre las mismas. Los detalles de esta comunicación se darán en el capítulo cuarto de esta memoria.

2.5.2 Wiris

El software cedido por MathForMore consta de varios módulos. Fue cedido como una API (Aplication Program Interface), empaquetado en un único archivo.

Este software consta de varias utilidades, siendo todas variaciones de una interfaz principal, que permite la creación de forma gráfica de fórmulas matemáticas. El módulo principal consta de un área general de edición y una barra de herramientas, que incluyen varias pestañas con los elementos que se pueden elegir para componer la fórmula.

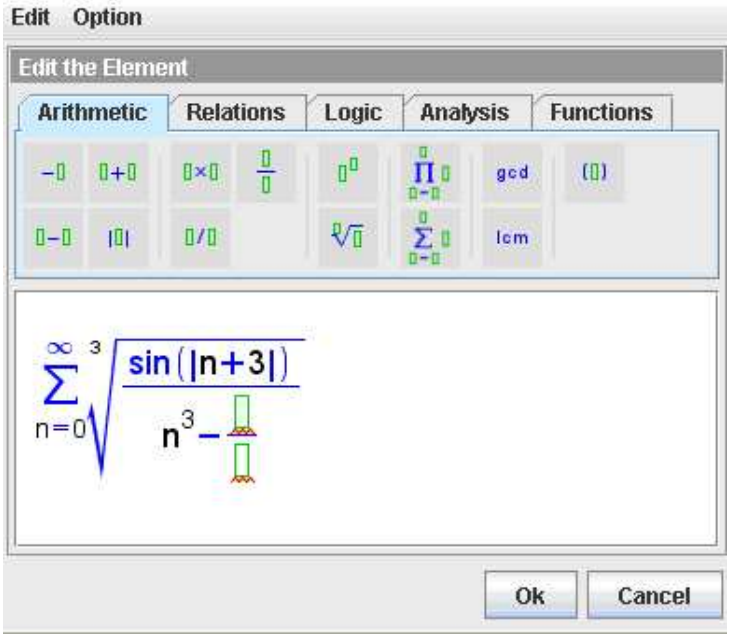


Figura 2.24: Interfaz principal de Wiris

Estas pestañas son configurables, dependiendo del contexto en el que se cargue la aplicación. Este contexto depende de unos archivos XML [20] que contienen los diferentes dominios matemáticos que incluye cada pestaña.

Esta interfaz principal se puede configurar a través de varios parámetros, pudiendo por ejemplo, desactivar la opción de la comprobación de la sintaxis (esto es, el subrayado en sierra rojo) o por ejemplo, la no aparición de menús. Todos los parámetros estaban especificados en la documentación que se proporcionó junto a la API.

Entre las variaciones de la interfaz principal que se encuentran en la API, encontramos una muy importante que es la posibilidad de mostrar el área de edición sóla, y lanzar el editor haciendo clic con el ratón de la fórmula encuadrada en color rosa. También cabe la posibilidad de hacerla no editable.

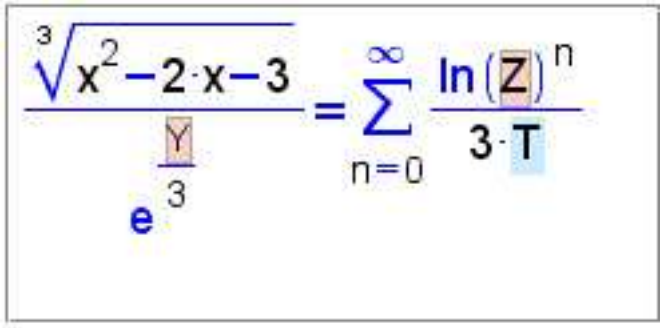


Figura 2.25: Variación de la interfaz principal de Wiris

Todas estas variaciones trabajan con OpenMath como estándar para la entrada y salida. Internamente, para mostrar gráficamente las fórmulas, se trabaja con MathML, aunque las fuentes que se usan también son configurables. Sobre OpenMath y MathML se hablará detalladamente en el siguiente capítulo.

Capítulo 3
Tecnologías Usadas

3 Tecnologías usadas

Tanto para el desarrollo y la implementación del proyecto y la realización de la memoria se han usado una gran variedad de diferentes tecnologías. Cada una de ellas ha sido usada en su ámbito por alguna razón específica y teniendo en cuenta el gran abanico de alternativas posibles.

Podemos clasificarlas en dos grandes grupos, estos son, las tecnologías usadas para la programación y la codificación, y aquellas que se han usado para la representación de la información. Dentro del primer grupo se distinguen entre tecnologías empleadas en el lado del servidor o en el lado del cliente.

De cada una de ellas haremos una breve introducción, la descripción del ámbito en el que se usa normalmente, su uso dentro del proyecto y su elección frente a otras alternativas (caso de que hubiera).

3.1 Programación

Dividimos esta sección las tecnologías usadas para la programación, construcción y despliegue de todo el software que contempla el proyecto en tres categorías: tecnologías de servidor, tecnologías de cliente y otras.

Como tecnologías de servidor nos referimos a aquellas que han sido necesarias para concretar la parte del proyecto que cae en la parte del servidor de una arquitectura cliente / servidor como es SIETTE. Las que incluimos en la categoría de cliente son aquellas que se utilizan en el lado del cliente (esto es, los navegadores web). Las que describimos en la 'otras' son aquellas que fueron necesarias durante el proyecto, pero que no entran en este esquema cliente / servidor.

3.1.1 Tecnologías de servidor

JAVA

Java es un lenguaje orientado a objetos desarrollado por James Gosling (entre otros) en Sun Microsystems a comienzo de los 90 [10]. Al contrario que otros lenguajes de programación convencionales, que están normalmente diseñados para ser compilados

a código máquina nativo o para ser interpretados en tiempo de ejecución, Java está diseñado para ser compilado a un código intermedio que luego es ejecutado en la máquina virtual de Java. Esta máquina virtual sí que es dependiente de la plataforma y asegura que cualquier programa escrito en Java pueda ejecutarse en cualquier plataforma que disponga de esta máquina virtual.

El lenguaje en sí es muy parecido sintácticamente a C o C++, pero tiene un modelo de objetos más simple y ofrece menos funcionalidades a bajo nivel. La sintaxis actual ha cambiado muy poco con respecto a sus comienzos, pero lo que ha sufrido un increíble crecimiento es plataforma de programación de calidad que ha crecido alrededor de él:

Java Platform, Enterprise Edition, o JAVA EE (J2EE desde la versión 1.4) [11] es una plataforma de programación para desarrollar y ejecutar aplicaciones de arquitectura multi-capas distribuidas, basadas ampliamente en componentes software modulares que se ejecutan en servidores de aplicaciones.

Esto permite al desarrollador crear aplicaciones empresariales que son portables entre plataformas y escalables, a la vez que se puede integrar con las tecnologías ya existentes.

SIETTE es una aplicación creada con J2EE. El uso de Java dentro de este proyecto es por tanto obligatoria, al tratarse éste de una ampliación de las funcionalidades iniciales. Existen, no obstante, otras plataformas de desarrollo de aplicaciones, como .NET, de Microsoft, o la reciente SAP, en las que se programa con C, C++ o C# y ABAP, respectivamente.

Java y J2EE se han utilizado como lenguaje de programación y plataforma de desarrollo en gran parte de este proyecto. Más específicamente, el uso de Java Applets o de Java Server Pages ha estado muy presente, pero, pese a pertenecer a esta plataforma, se detallarán más adelante.

JSP (Java Server Pages)

Java Sever Pages, o Páginas de Servidor de Java es una tecnología de Java que permite generar dinámicamente HTML [20], XML [20] o otros tipos de documentos en respuesta a una petición Web de un cliente. Permite, por lo tanto, insertar código Java y ciertas acciones predefinidas un contenido que originalmente sería estático.

Estas páginas son luego compiladas y desplegadas como Servlets de Java, que son los que realmente responderán a la petición del cliente.

En el proyecto, JSP es usado en la integración de los Applets de Edición de enunciados dentro de SIETTE, ya que estos son llamados desde una página web (HTML) que ha sido generada dinámicamente dependiendo de los parámetros de SIETTE en el momento de la llamada.

La alternativa de Microsoft .NET para JSP es ASP, o Active Server Pages.

SQL

SQL es un lenguaje no sólo de consulta de Base de datos, sino que también puede definir la estructura de estos datos, modificarlos y especificar restricciones de seguridad [9].

Fue inicialmente desarrollado por IBM y ha evolucionado hasta convertirse en un estándar ANSI [28] e ISO [29]. No obstante, este estándar es muy amplio, y las diferentes implementaciones (Oracle, IBM, Microsoft, MySql …) dan lugar divergencias en aspectos de relativa importancia (fechas, horas, secuencias…) que provocan que la portabilidad sea prácticamente imposible sin realizar cambios importantes.

En SIETTE se soportan las versiones de SQL para Oracle y MySql. Para poder introducir el nuevo patrón y el nuevo editor hubo que añadir nuevos datos a la base de datos.

Tomcat

Apache Tomcat [41] es un contenedor de aplicaciones desarrollado por Apache Software Foundation. Implementa las especificaciones de servlets y JSP de Sun

Microsystems, proveyendo un entorno para ejecutar código Java en cooperación con un servidor web. Se configura con archivos XML.

SIETTE se despliega en un servidor Tomcat 4.1.31.

3.1.2 Tecnologías de cliente

HTML

HyperText Markup Language, es el principal lenguaje de marcado para la creación de páginas web. Provee un medio de describir la estructura de una información basada en texto en un documento, denotando parte del texto como cabeceras, listas, párrafos y demás y además, complementar este texto con formularios interactivos, imágenes y otros objetos. Igualmente también describe, hasta cierto punto, la apariencia y la semántica del documento, y puede facilitar la inclusión de código escrito en lenguajes de script, para modificar el comportamiento de los navegadores web.

HTML es usado extensivamente en SIETTE. Dado que es una aplicación basada en el modelo cliente / servidor, se sirve de páginas web que son renderizadas por los navegadores web. Los enunciados de las preguntas de SIETTE y las respuestas están escritos la gran mayoría en HTML. Por lo tanto, para el proyecto, y en concreto para el editor de enunciados se ha implementado para su uso con HTML.

JavaScript

JavaScript es un lenguaje de script basado en el estándar ECMAScript. Desarrollado inicialmente por Netscape, esta vagamente relacionado con Java. La sintaxis es parecida, ya que también recuerda a la de C. Es un lenguaje basado en los prototipos, que pese a que puede ser usado para acceder a los objetos de otras aplicaciones, es usado intensivamente en las páginas web para modificar el comportamiento de los navegadores.

El código puede ser escrito directamente dentro del documento HTML o dentro de ficheros que serán enlazados finalmente por el documento HTML

SIETTE basa la comunicación del HTML y los applets de edición y de respuesta en JavaScript, así como gran parte del comportamiento añadido a las páginas web.

CSS

Cascade Style Sheets es un lenguaje de hojas de estilos para describir la presentación de un documento escrito en un lenguaje de marcado. Su aplicación más común es para dar estilo a páginas web escritas en HTML o XHTML, pero se puede aplicar a cualquier tipo de documento XML, como SVG [44] o XUL [45]. Las especificaciones de CSS las establece el World Wide Web Consortioum [20].

Existen varios niveles, CSS1, CSS2 y CSS3. Cada uno añade funcionalidades al anterior.

Ya que CSS complementa HTML para su presentación, se tuvo en cuenta para la elaboración del editor de enunciados.

Java Applet

Los applets de Java son pequeñas aplicaciones que se ejecutan en los servidores web. Necesitan que la máquina virtual de Java esté operativa en el ordenador del cliente. Añaden funcionalidad a las páginas web y pueden comunicarse con el servidor.

Los applets se utilizan para incluir en una página web funcionalidad que no se puede conseguir con los lenguajes de script o el propio HTML, tales como la edición avanzada de textos, gráficos, juegos u otras pequeñas aplicaciones.

SIETTE se apoya en applets de edición tanto generales como específicos para editar en línea los enunciados y las respuestas. Existen applets para algunos ítems específicos además.

Wiris [16], el software cedido por MathForMore implementa una pequeña batería de applets para el uso de su software en clientes web.

3.1.3 Otras

Ant

Apache Ant [40] es una herramienta software para automatizar los procesos de compilación y construcción del software. Es similar a 'make' pero está escrito en Java y está principalmente orientado a su uso con Java.

Una de las principales diferencias entre ant y make es que el archivo donde se describe es un archivo XML y no uno de formato Makefile. En este archivo se describen las tareas con unos tags predefinidos por ant. Permite establecer prioridades, dependencias, y puede realizar tareas básicas como compilación, copiar o crear archivos, o tareas más complejas como la empaquetación de aplicaciones.

Dentro del proyecto SIETTE, las tareas de construcción y despliegue se desarrollan con ant a través de ficheros XML. Para este proyecto hubo que modificar estos ficheros para incluir las nuevas funcionalidades añadidas.

CVS

Concurrent Version System [42], o Sistema Concurrente de Versiones, implementa un sistema de control de versiones, esto es, mantiene todo el trabajo y los cambios producidos en un conjunto de ficheros, normalmente un proyecto software.

Este sistema permite a varios compañeros de un mismo proyecto (y potencialmente separados) trabajar en un mismo proyecto y poder ver y compartir los cambios hechos.

En el desarrollo de SIETTE se utiliza un repositorio CVS, y todos los archivos del proyecto también formaron parte de éste.

3.2 Representación de la información

Estas tecnologías, a diferencia de las usadas para programación, se emplean dentro del proyecto para representar la información, no para tratarla. Hablaremos de OpenMath y MathML para la representación de fórmulas matemáticas, de XML, que engloba a las anteriores y es la gran estrella de la informática de los últimos tiempos. Por último, describiremos UML y su uso dentro de esta memoria.

XML

XML, el Lenguaje de Marcado Extensible (eXtensible Markup Language), es un estándar de W3C para marcado de documentos. Define una sintaxis genérica que se usa para marcar datos con etiquetas simples, leíbles por humanos. Provee por lo tanto, de un formato estándar para documentos para ordenadores. Sin embargo, este formato es suficientemente flexible para poder acomodarlo dominios tan diversos como sitios web, intercambio electrónico de datos, gráficos vectoriales, música, matemáticas, serialización de objetos o llamadas remotas [12].

Los datos se incluyen en XML como cadenas de texto, y luego, estos datos se rodean de etiquetas de marcado que describe los datos. La propia especificación de XML define cómo deben de ir marcados estos datos, que restricciones existen (tales como tipo de anidamientos, atributos, etc.…). Lo más importante es que XML es un meta-lenguaje de marcado. Es decir, que no tiene un conjunto fijo de etiquetas de marcado, sino que permite al usuario crear las etiquetas conforme las vaya necesitando. Estas se especifican a través del DTD (Document Type Definition) o, a través de los XML Schemas, siendo esta última la manera más eficiente y con más posibilidades, Por eso se dice que es extensible.

El uso de XML se ha extendido enormemente los últimos 5 años, y ahora se puede encontrar en un sinfín de aplicaciones informáticas. Su portabilidad, y el amplio crecimiento del número de herramientas (y de su calidad) que trabajan con él han contribuido a esto.

Un ejemplo de documento XML sencillo podría ser el que ofrece W3C en su web:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note  [
      <!ELEMENT note (to, from, heading, body)>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body (#PCDATA)>
]>
<note>
      <to>Tove</to>
```

```
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
</note>
```

Se puede observar que se inicia con una declaración de que es un documento XML, la especificación en forma de DTD interno, y a continuación el documento en sí, con texto marcado con etiquetas.

Un uso muy común de un documento XML es la configuración de una aplicación. A través de XML, se puede crear a medida los datos de entrada que necesita una aplicación para los ajustes internos que necesite.

Dentro del proyecto, éste es el caso de los applets de WIRIS, que configuran su dominio inicial a través de ficheros XML.

MathML

MathML es un lenguaje de marcado basado en XML, cuyo objetivo es expresar notación matemática de forma que distintas máquinas puedan entenderla, para su uso en combinación con XHTML en páginas web, y para intercambio de información entre programas de tipo matemático en general.

MathML, al estar basado en XML, define unas etiquetas para marcar texto, a la vez que

El soporte de MathML es bastante extenso en programas matemáticos como Maple o Mathcad, pero aún es escaso en los navegadores web, por ejemplo.

La siguiente fórmula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

se puede expresar en MathML con el siguiente código:

```
<math>
 <mi>x</mi>
 <mo>=</mo>
```

```
 <mfrac>
  <mrow>
   <mrow>
    <mo>-</mo>
    <mi>b</mi>
   </mrow>
   <mo>&PlusMinus;</mo>
   <msqrt>
    <msup>
     <mi>b</mi>
     <mn>2</mn>
    </msup>
    <mo>-</mo>
    <mrow>
     <mn>4</mn>
     <mo>&InvisibleTimes;</mo>
     <mi>a</mi>
     <mo>&InvisibleTimes;</mo>
     <mi>c</mi>
    </mrow>
   </msqrt>
  </mrow>
  <mrow>
   <mn>2</mn>
   <mo>&InvisibleTimes;</mo>
   <mi>a</mi>
  </mrow>
 </mfrac>
</math>
```

Dentro del proyecto, el uso de MathML se restringe a la herramienta WIRIS, que lo utiliza como lenguaje interno de representación para su software.

OpenMath

OpenMath [5] es un estándar emergente para la representación de objetos matemáticos junto a su semántica. Esto permite que sean intercambiables entre aplicaciones, guardados en bases de datos o publicados en la web.

OpenMath está basado en XML, por lo cual, es extensible. Consiste en la definición de objetos matemáticos y en la definición de "Content Dictionaries" [17], o diccionarios de contenidos, que especifican los conceptos matemáticos con los que se formarán los objetos OpenMath. Estos diccionarios de contenidos son extensibles, y pueden dar cabida a los más diversos conceptos de las matemáticas. Contemplan tanto la representación de estos objetos como su semántica, sea forma o informal.

OpenMath está diseñado para soportar MathML como parte interna de su especificación, y estas dos tecnologías son altamente complementarias.

La comunidad OpenMath también define "Phrasebooks", que son pequeñas piezas de software que traducen entre objetos OpenMath y el lenguaje nativo de estas.

De este modo, teniendo los Phrasebooks adecuados, se puede crear una comunicación entre dos aplicaciones matemáticas que, inicialmente, no compartían la misma representación interna de los objetos matemáticos con los que trabajaban [24]. El siguiente esquema reproduce esta situación:

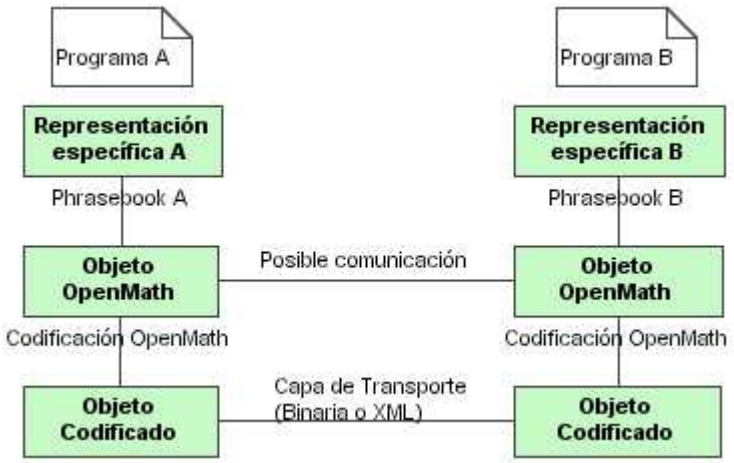
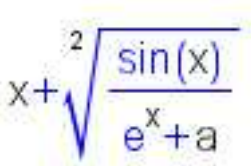


Figura 3.1: Intercambio de objetos Openmath

La siguiente fórmula:



se puede expresar en OpenMath con el siguiente código:

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0"
cdbase="http://www.openmath.org/cd">
   <OMA>
      <OMS cd="arith1" name="plus"/>
      <OMV name="x"/>
      <OMA>
         <OMS cd="arith1" name="root"/>
         <OMA>
            <OMS cd="arith1" name="divide"/>
            <OMA>
               <OMS cd="transc1" name="sin"/>
```

```
            <OMV name="x"/>
         </OMA>
         <OMA>
            <OMS cd="arith1" name="plus"/>
            <OMA>
               <OMS cd="transc1" name="exp"/>
               <OMV name="x"/>
            </OMA>
            <OMA name="a"/>
         </OMA>
      </OMA>
      <OMI>2</OMI>
   </OMA>
</OMA>
</OMOBJ>
```

Dentro del proyecto, el uso de OpenMath es extensivo, dado que todas las fórmulas, sean respuestas, parte de enunciados de preguntas o ejemplos, están representadas con esta tecnología.

UML

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está apoyado en gran manera por el OMG [46] (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables [15].

El punto importante a notar es que UML es un "lenguaje" para especificar y no un método o un proceso. UML se usa para definir un sistema de software; para detallar los artefactos en el sistema; para documentar y construir -es el lenguaje en el que está descrito el modelo. UML se puede usar en una gran variedad de formas para soportar una metodología de desarrollo de software (tal como el Proceso Unificado de Rational) -pero no especifica en sí mismo qué metodología o proceso usar.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. Estos diagramas abarcan todo el ámbito de una

aplicación, incluyendo diagramas de clases, de colaboración entre componentes, de estructuras, de estados, de actividades, de casos de uso o de implementación.

Dentro del proyecto, UML se ha usado para especificar claramente el sistema, durante el proceso software y sobretodo en esta memoria.

Capítulo 4
Procesos Software

4 Procesos Software

Hay diferentes métodos de creación de un producto software hoy en día. Desde el obsoleto desarrollo en cascada, a los procesos ágiles, pasando por la programación extrema.

En este proyecto se ha seguido una metodología de desarrollo en espiral. Inicialmente se va creando una pequeña parte del sistema, y manteniéndolo siempre funcional, en cada iteración del proceso se van añadiendo nuevas funcionalidades que van haciendo crecer el mismo. Por lo tanto, siempre hay un prototipo funcional que sirve como base para la siguiente iteración, y de cuyas pruebas se obtienen nuevos requisitos que completan la especificación del mismo.

Dentro de cada iteración se pueden distinguir cuatro aspectos bien diferenciados del desarrollo:

Primero está el análisis de requisitos, esto es, el entendimiento de las funcionalidades que se van a implementar, su abstracción a pequeños requerimientos independientes con los que se pueda ir construyendo las funciones deseadas. Esta parte es más importante al comienzo del proyecto, donde están menos claras las funcionalidades finales, ya que pueden variar, y bastante durante todo el proceso. Todos estos requerimientos quedan especificados en casos de uso, o sea, escenarios de uso del software, y con tablas de requisitos.

Luego está la fase de diseño. A través de los requerimientos obtenidos en la fase anterior, se diseña una arquitectura para el sistema, se detallan los comportamientos de las partes del mismo y las relaciones entre ellas. Se especifican las secuencias de interacciones entre los componentes, las evoluciones del estado de cada parte que sea susceptible de cambiar durante el uso del sistema. A través de diagramas de clases, de interacciones, de secuencias o de estados, se detallan lo que luego será la implementación. Importante también, notar el uso de los patrones de diseño, o pequeñas soluciones a problemas recurrentes que son muy útiles pues ahorra tiempo y mejoran la eficiencia del software.

La fase de implementación es la tercera, que dado el diseño y los requisitos, y el (o los) lenguajes de implementación que se van a usar es la que realmente se encarga de reflejar todo el comportamiento ya especificado en las fases anteriores.

Por último, la fase de prueba e integración, donde se testean que los resultados esperados en la fase de análisis son los que realmente se han conseguido tras la implementación y se integra dentro del entorno.

Obviamente, no en todas las iteraciones estas fases tienen la misma importancia. Al comienzo del proyecto hay mucho más análisis que al final, mientras que, por ejemplo, las pruebas ocupan gran parte de las últimas iteraciones del proyecto, y el diseño y la implementación suelen permanecer constantes. El siguiente gráfico explica el proceso seguido:

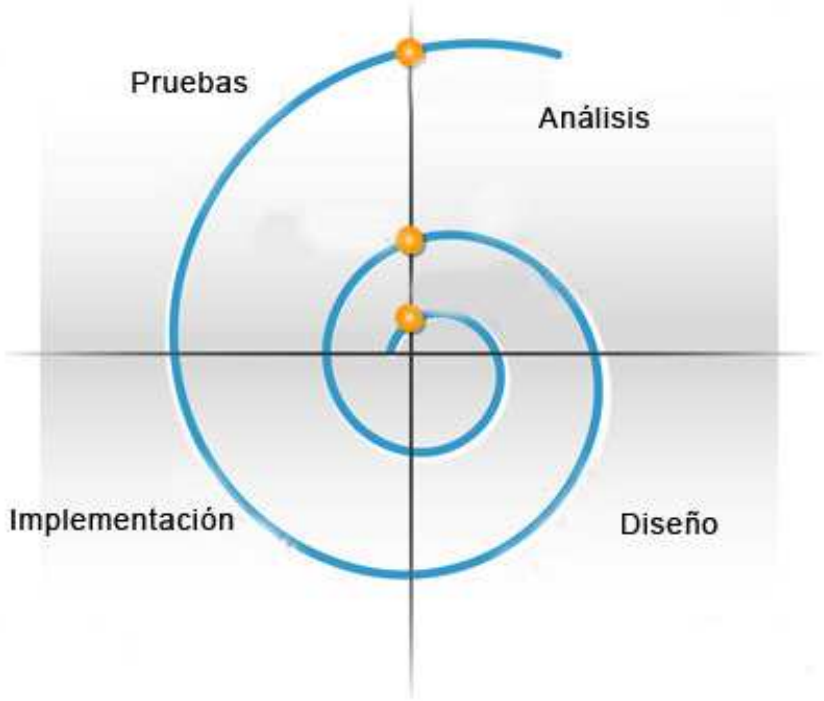


Figura 4.1: Proceso incremental e iterativo

Para este sistema se ha dividido claramente las iteraciones en la secuencia de las funcionalidades deseadas. Inicialmente, se buscaba que SIETTE permitiera usar un nuevo tipo de patrón, que reconociera objetos OpenMath y los comparara con aquellos dados por el profesor. Luego habría que añadir la posibilidad de que el alumno no tuviera que introducir un código OpenMath para responder, de ahí el uso de un applet de

edición gráfica de respuestas. Implementar la comunicación entre este applet y el nuevo patrón, y permitir que el alumno pudiera ver su respuesta de manera gráfica, y no a través de código OpenMath. Finalmente habría que facilitar la tarea del profesor para la creación de enunciados complejos, compuestos de código HTML formateado y fórmulas matemáticas y que no tuviera por qué saber por ello HTML o los pasos necesarios para poder mostrar gráficamente una fórmula usando los applets de WIRIS. También se facilita la edición de respuestas, sin necesidad de conocer OpenMath. Además, habría que probar el sistema, para ello se crea la batería de tests y preguntas. Esto se detalla en el quinto capítulo.

Por lo tanto, podemos dividir el este capítulo en estos cuatro grandes bloques, Análisis, Diseño, Implementación y Pruebas. De cada uno de estos grandes bloques también creamos una división en base a las funcionalidades que se han comentado: el nuevo patrón OpenMath, que denominaremos Evaluador, la Comunicación del Alumno con el Test, y los Applets de Edición, tanto de Enunciados como de Respuestas.

4.1 Análisis

Para distinguir los diferentes requisitos que se enumeran en esta fase, haremos la clasificación que indica Roger Pressman [13]. Requisitos de Datos de entrada, de datos de Salida, de Comportamiento, de Funcionalidad y de Interfaz (representados por las letras D, S, C, F e I respectivamente). Los dos primeros tipos son claros, indican requisitos de los datos de la entrada y de la salida. Los de comportamiento indican el funcionamiento deseado de alguna parte del software mientras que los de funcionalidad indican más bien una funcionalidad abstracta que un comportamiento. Los de interfaz indican requisitos de la interfaz de usuario.

4.1.1 Evaluador

La misión principal del evaluador es recibir objetos OpenMath y compararlos con los establecidos con el profesor para determinar si el alumno ha respondido correctamente.

Esto plantea el problema de cómo saber si dos fórmulas matemáticas son iguales. En las definiciones vistas en el capítulo dos, distinguíamos entre igualdad

semántica e igualdad sintáctica. La fórmula x+3 puede ser igual a x^2/x+3 o a x+4-1. Obviamente, comparando sintácticamente estas fórmulas, la posibilidad de un falso negativo es muy alta, ya que tanto el profesor como el alumno podrían encontrar fácilmente dos maneras alternativas de escribir el mismo resultado. Por lo tanto un requisito principal del evaluador es que utilice como criterio de comparación la igualdad semántica, hasta el punto que le sea posible.

Dado que se puede conseguir un amplio espectro de fórmulas matemáticas combinando los operadores simples y algunas funciones como los logaritmos, exponenciales o trigonométricas, podemos restringir el dominio de las respuestas del alumno y las propuestas por el profesor a las siguientes:

Operadores matemáticos simples: mas, menos, menos unario, división, multiplicación, raíz n-aria y exponenciación.

Funciones: absoluto, logaritmo neperiano, logaritmo en base n-aria, exponencial, seno, coseno, tangente, y sus inversas, arco seno, arco coseno, arco tangente y sus inversas.

Como se explicará en el capítulo quinto, la restricción del dominio a estas funciones no merma en absoluto la capacidad de crear una gran variedad de ejercicios.

Una vez establecido que hay que buscar una igualdad semántica entre respuestas, se plantea cómo ha de lograrse esto. Se hace mediante una evaluación numérica que compruebe si ambas fórmulas dan el mismo resultado para las mismas instanciaciones de sus variables (si ha lugar). Esta evaluación ha de hacerse un número determinado de veces, y con un margen de error.

Además, éste evaluador ha de contemplar la posibilidad de que la evaluación podría usar valores que estuvieran fuera del dominio de la fórmula, y produjera indeterminaciones.

En caso de que todas las evaluaciones de las fórmulas produjeran indeterminaciones, el sistema, en vez de fallar, debe degradar su comportamiento y

basarse simplemente en la igualdad sintáctica, que utilizará como última baza para comprobar la corrección de las respuestas.

Por otro lado, el nuevo patrón ha de integrarse completamente en la batería de patrones que ya dispone SIETTE, y presentar el mismo comportamiento que el resto. Además, dado que las respuestas han de evaluarse en tiempo real, para muchos alumnos, es deseable que sea un proceso ligero.

Resumiendo, obtenemos los siguientes requisitos:

Tipo	Requisito
Funcionalidad	
	Comprobar la corrección de una respuesta del alumno
Comportamiento	
	Hacer un número determinado de evaluaciones.(10)
	Comprobar la igualdad sintáctica en último caso si falla la igualdad semántica.
	Hacer evaluaciones de manera que no requieran una gran carga de proceso.
Datos de entrada	
	Una fórmula de respuesta del alumno y una o varias de las del profesor.
Datos de salida	
	Una serie de valores booleanos por cada respuesta del profesor, indicando si es igual a la del alumno o no.

Tabla 4.1: Requisitos Evaluador

4.1.2 Comunicación Alumno Test

Lo deseable de un sistema de entorno web de tests de cálculo con preguntas de respuesta libre es que el alumno pueda contestar a las preguntas planteadas con algún método sencillo y gráfico de representación de objetos matemáticos. A través de texto plano esto es casi imposible cuando intentamos representar algo más que simples operaciones aritméticas. También es importante que el alumno pueda ver si su respuesta era acertada o no, y por lo tanto, hay que preservar de alguna manera esa representación gráfica a través de todo el proceso de la pregunta.

A través de los applets que ofrece la herramienta Wiris se soluciona el problema de poder escribir fácilmente una fórmula matemática. El problema es que estos applets devuelven código OpenMath, que aún siendo la respuesta del alumno, no se puede

mostrar más adelante dentro de un documento HTML para ser representado en un entorno web. El comportamiento deseado en este caso sería el que muestra el siguiente diagrama de interacción:

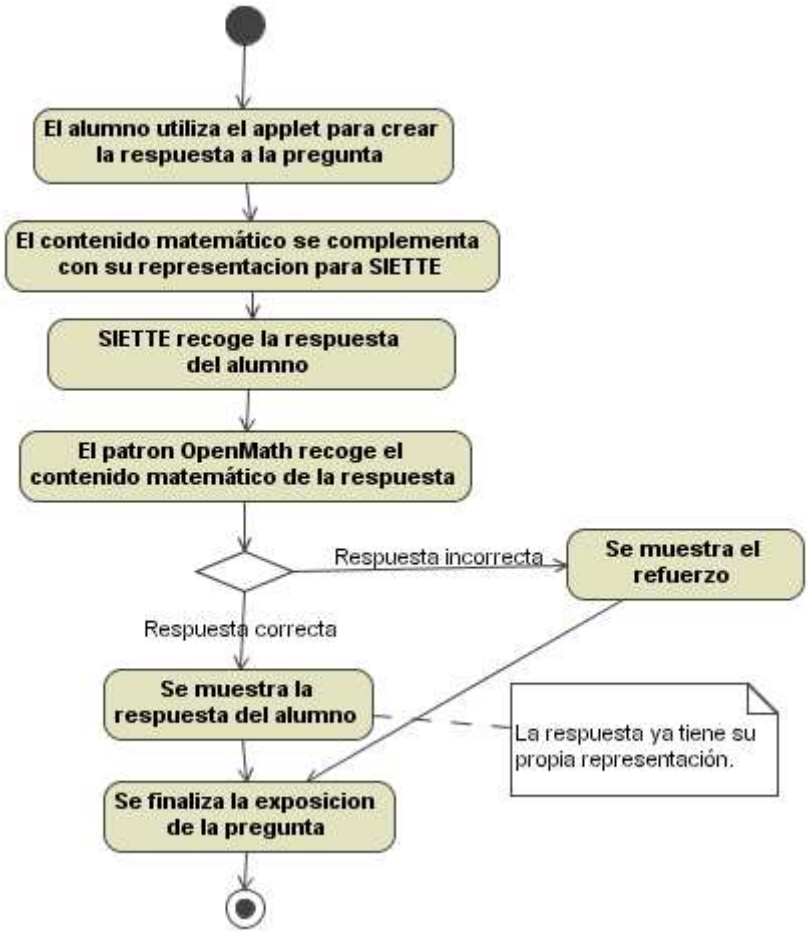


Figura 4.2: Diagrama de interacción alumno-respuesta

Por lo tanto, uno de los requisitos de esta fase es adaptar los applets que ofrece Wiris a que devuelvan el contenido matemático de la respuesta envueltos por su propia representación gráfica, y que luego, una vez lleguen las respuestas al servidor, se obtenga fácilmente el contenido matemático de éstas.

Otro requisito importante es integrar el applet que el alumno utiliza para responder dentro del sistema que tiene SIETTE para interactuar con los applets de respuestas en las preguntas. Por lo tanto, el comportamiento que se espera de este applet es el siguiente:

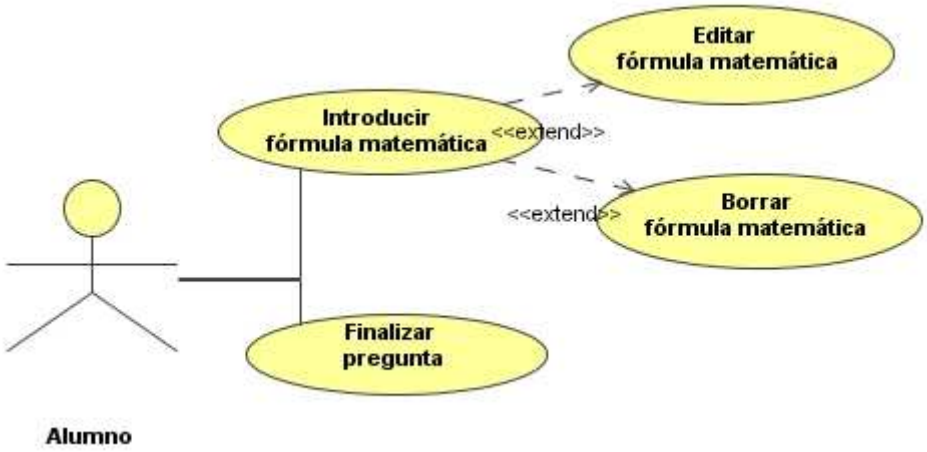


Figura 4.3: Diagramas de casos de uso del applet de respuestas

Los tres primeros casos de uso están implementados por el propio applet de Wiris. Es el cuarto, Finalizar pregunta, el nuevo que hay que añadir.

Para resumir, estos son los requisitos principales de esta fase:

Tipo	Requisito
Funcionalidad	
	Ampliar la funcionalidad del applet de Wiris para que interactúe con SIETTE.
Comportamiento	
	Hacer que el applet de Wiris devuelva una respuesta que contenga el código OpenMath de la respuesta dentro de su representación gráfica.
	En el servidor, recoger la respuesta y obtener el código OpenMath para pasárselo al patrón OpenMath

Tabla 4.2: Requisitos Comunicación Alumno-Test

4.1.3 Applets de Edición

Los applets de edición que se han realizado en este proyecto son dos. Uno para los enunciados hechos en HTML que puedan contener fórmulas matemáticas representadas de forma gráfica a través de los applets de Wiris y otro para editar las respuestas que da el profesor para una pregunta y que están expresadas en código OpenMath.

Ambos applets han de integrarse junto a la batería de applets de edición que ya posee SIETTE (véase apartado 2.5, Estado previo del problema), utilizando los mismos

métodos que utiliza el resto para ser lanzados como editores externos desde cualquiera de los entornos de edición de preguntas que posee SIETTE (TEDI y AthoS).

Los editores externos en Siette son todos applets de Java, que son llamados desde código HTML. Recogen el texto que han de editar desde el entorno de edición donde estén siendo llamados y lo procesan para mostrarlo de la forma gráfica adecuada. Casi todos son editores de propósito específico, como vimos. Hay un editor para enunciados SQL, otros para enunciados referentes a la asignatura de Procesadores de Lenguajes, etc. … Hermes es un editor de propósito general que, sin embargo, obliga a la posesión de conocimientos de HTML básicos, ya que no analiza el texto para mostrarlo ya formateado (aunque tenga un módulo que haga esto pero sin posibilidad de edición). Una de los requisitos básicos es que estos editores sigan las líneas de una interfaz WYSIWYG (“What You See Is What You Get”, es decir “lo que ves es lo que obtienes”), no obligando al usuario a tener que comprender la representación interna de lo que está editando.

4.1.3.1 Applet de Edición de Enunciados

Ya hemos establecido dos requisitos importantes. Que el editor se integre dentro de Siette de la misma manera que el resto de editores y que siga una línea de desarrollo que permita el objetivo WYSIWYG.

Principalmente lo que se busca es un editor de textos, que lea HTML, y analizándolo lo muestre de forma visual en un entorno de edición. Es decir, que, por ejemplo, cuando se encuentre en el texto algo que indique que una parte del texto ha de ir en negrita, muestre en el entorno de edición esa parte del texto en negrita. Y cuando desde el editor de tests se le pida que devuelva el texto, que devuelva la representación interna en HTML.

Vimos en la sección 3 de esta memoria, que hay diferentes versiones del lenguaje HTML. La más reciente, HTML 4.0 ha dejado obsoleto algunos aspectos de las anteriores, y vimos que ciertos elementos ya no están permitidos en esta última versión. Por eso el editor deberá aceptar todas las versiones anteriores a la 4.0 (incluyendo ésta, claro está) y convertir los textos que edite a la más actual, modernizando así los contenidos y adaptándolos a que puedan ser mostrados en todos

los navegadores modernos. La lista completa de elementos que admite este editor se encuentra en los anexos.

Una vez analizado el texto, el usuario podrá añadir más texto, con o sin formatos especiales, borrar o cambiar el formato a texto ya existente resaltándolo. Los formatos que se podrán aplicar serán cursiva, negrita, subrayado, color (rojo, azul y negro), y alineación (centrada, a la derecha, a la izquierda y justificada).

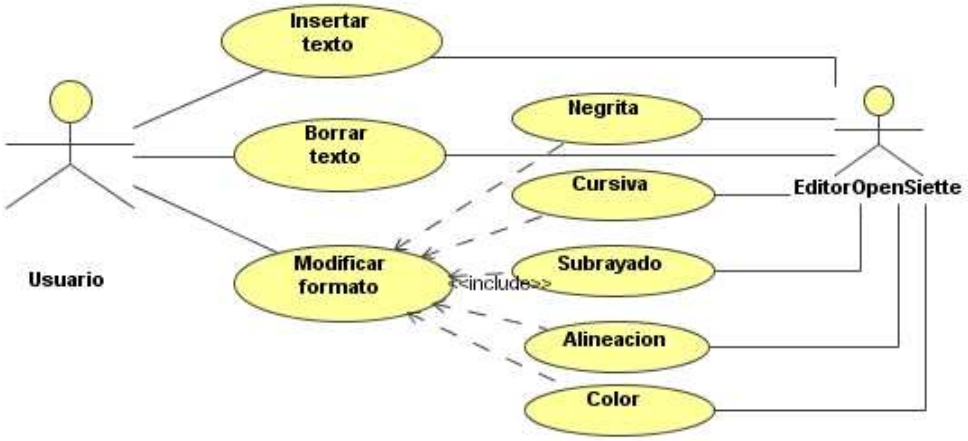


Figura 4.4: Diagrama de casos de uso EditorOpenSiette (I)

Para esto, la interfaz deberá incluir todos los botones necesarios para estos formatos. También la interfaz permitirá resaltar los textos e incluir texto en cualquier punto de ello.

Otra característica importante de un editor de textos es que permita hacer y deshacer las últimas acciones realizadas. Habrá que implementar esta característica e incluir los botones necesarios para ello.

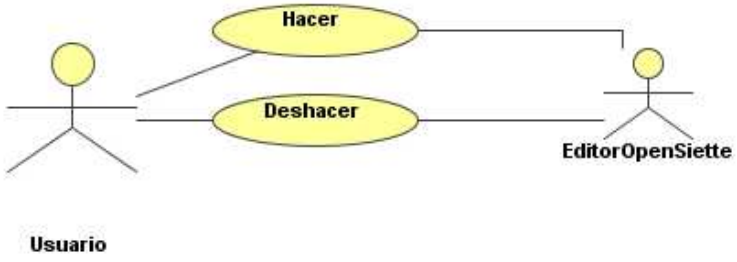


Figura 4.5: Diagrama casos de uso EditorOpenSiette(II)

Muy importante, queremos que el editor detecte la representación de una fórmula matemática en HTML (que como hemos visto será como en la comunicación alumno –test, un applet de HTML conteniendo el código OpenMath de la fórmula) y la represente gráficamente con una fórmula matemática que sea editable. Cuando se haga clic sobre esta fórmula aparecerá un editor (extraído de la API de Wiris) específico en el cual se editará la fórmula. Por supuesto, esta fórmula también será tratada internamente como HTML, y cuando se le pida al editor devolver el código que representa el texto editado, devolverá el código HTML correspondiente a la misma (tal como se hace en la comunicación alumno test).

Por último, hay que incluir en el editor un botón que permita añadir al enunciado el applet que el alumno utiliza para responder.

Resumiendo, tenemos los siguientes requisitos:

Tipo	Requisito
Funcionalidad	
	Edición del texto de entrada
	Interpretación del HTML del texto de entrada
Comportamiento	
	Mostrar la vista del texto (WYSIWYG)
	Permitir eliminar texto
	Permitir insertar texto
	Permitir resaltar texto y aplicar texto
	Hacer / Deshacer
	Formatear el texto con negrita/cursiva/subrayado
	Formatear el texto con colores (negro, rojo, azul)
	Formatear la alineación del párrafo (centrado, derecha, izquierda o justificado)
	Permitir la inclusión de nuevas fórmulas matemáticas
	Edición de una fórmula haciendo clic sobre ella
Datos de entrada	
	Texto HTML del enunciado
	Las fórmulas matemáticas están representadas con HTML
Datos de salida	
	Texto HTML optimizado
	Las fórmulas matemáticas cambiadas o añadidas están representadas con HTML
Interfaz	
	Integración con los Editores de Tests de Siette (TEDI y AthoS)

	Botones para negrita / cursiva / subrayado
	Botones para alineación del párrafo (centrado, derecha, izquierda, justificado)
	Botones para el color (rojo, negro, azul)
	Botones para hacer / deshacer
	Botón para incluir nuevas fórmulas matemáticas

Tabla 4.3: Requisitos EditorOpenSiette

4.1.3.2 Applet de Edición de Respuestas

El applet de edición de respuestas sería otro de los applets de edición externa a implementar dentro de este proyecto. Sirve para crear y editar los patrones de respuesta que los profesores definen en los ítems y que están expresados en código OpenMath.

Obviamente no podemos esperar que el usuario conozca el estándar OpenMath, por eso hemos de trasladar el código XML de la respuesta, interpretarlo y mostrarlo de forma gráfica. Una vez hecho esto, debemos dar la posibilidad de editar las fórmulas y posteriormente devolver el código OpenMath de lo que el usuario ha editado.

Este editor debe dar la opción de añadir aquellos elementos matemáticos que están implementados en el evaluador.

Por lo tanto estos son los requisitos de este editor:

Tipo	Requisito
Funcionalidad	
	Edición de la fórmula de entrada
	Interpretación del código OpenMath del texto de entrada
Comportamiento	
	Mostrar la vista de la fórmula (WYSIWYG)
	Permitir eliminar elementos a la fórmula
	Permitir insertar elementos de la fórmula
	Restringir el dominio del editor al del evaluador
Datos de entrada	
	Fórmula representada en código OpenMath
Datos de salida	
	Fórmula representada en código OpenMath
Interfaz	
	Integración con los Editores de Tests de Siette (TEDI y AthoS)
	Botones para todos los elementos del dominio

Tabla 4.4: Requisitos Editor Respuestas Profesor

4.2 Diseño

En la fase de diseño, siguiendo los requisitos especificados en la fase del análisis, se van modelando las diferentes clases e interfaces que luego serán programadas. Siguiendo las prácticas de los patrones de diseño, la modularización y los paradigmas de la programación a objetos, poco a poco se va creando la estructura del software ideado.

4.2.1 Evaluador

El evaluador, tal como se especificó en la parte de análisis, debe integrarse como un patrón más de la batería de patrones de SIETTE. La clase Patrón, dentro de SIETTE, es una clase abstracta que define una interfaz con un único método, *int pertenece(String st)*, que todo patrón debe implementar al heredar de la misma.

El proceso es el siguiente: el núcleo de SIETTE obtiene la respuesta del alumno. Al ser una pregunta de respuesta libre, busca el tipo de patrón de la pregunta, que es un atributo inherente a la misma. Dependiendo de éste, instancia el tipo de patrón correspondiente. Luego, por cada respuesta, invoca al método pertenece. Éste devuelve el índice del patrón que ha correspondido con la respuesta. Si devuelve -1, es que la respuesta era incorrecta para todos los patrones definidos por el profesor.

Por lo tanto, el nuevo patrón *OpenMathPattern,* debe comparar la respuesta que le llega al invocar el método *pertenece* con los patrones dados por el profesor. Esto se hace de la siguiente manera.

Como quedó especificado, la igualdad debe ser semántica. Tanto la respuesta del alumno como el patrón del profesor son fórmulas matemáticas expresadas en código OpenMath. Dado que está expresada en XML en forma de árbol binario (véase la sección 3.2 de esta memoria.), vamos a representarlo internamente como una estructura arbórea. Esto se explicará más detalladamente un poco más abajo. El diseño del patrón OpenMath depende por lo tanto de las evaluaciones de estas fórmulas. Por cada para respuesta/patrón se crea un comparador, que analizará el texto XML, creará la estructuras internas que representas las fórmulas OpenMath y las evaluará, devolviendo un valor booleano. La evaluación se hace dando valores aleatorios a las variables que

hayan podido surgir. Estas variables se almacenan en una clase común, llamada *Variables*, que implementa el patrón de diseño Singleton, es decir, que sólo presenta una instancia y es accesible desde cualquier punto del sistema.

El proceso de pasar de la representación en XML de la fórmula a una representación interna propia se hace a través de un analizador de XML. Este analizador implementa el patrón de diseño Oyente. Un analizador de lenguajes ascendente (para más detalles, véase el apartado 4.3.1) va leyendo el texto XML y cada vez que encuentra un token, lanza un evento que recoge el manejador de eventos, valga la redundancia. Este manejador va construyendo la representación interna en base a estos eventos.

El siguiente diagrama muestra lo descrito hasta el momento:

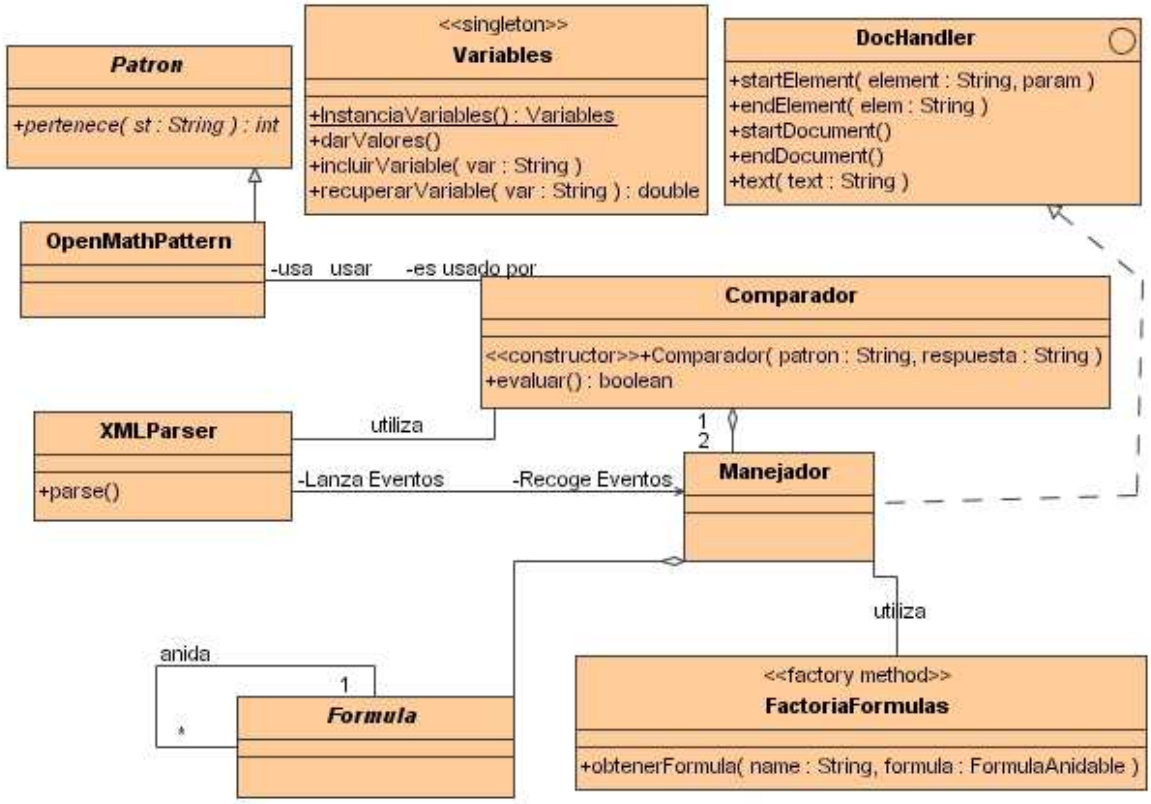


Figura 4.6: Diagrama de clases PatronOpenmath

Como se aprecia, la clase *Comparador*, utiliza dos clases *Manejador*, una para la respuesta del alumno y otra para el patrón del profesor. Éstas, a su vez, cuando termina

el análisis del texto XML, tienen una estructura arbórea de clases Fórmula. La interfaz *DocHandler* es la que especifica los métodos que recogen los eventos. Estos eventos son el inicio del documento, el final del mismo, el comienzo y fin de un elemento (como <OMA></OMA>) y la aparición de texto. La clase *FactoriaFormulas*, que implementa el patrón de diseño FactoryMethod, es una clase que aporta nitidez al código.

Sólo resta por comentar el diseño de la estructura arbórea en que se convierte cada fórmula OpenMath.

Una fórmula OpenMath siempre empieza por un elemento OMOBJ. Luego puede aparecer un elemento de Aplicación (OMA), o uno de estos tres: una variable (OMV), un número entero (OMI) o un número decimal (OMF). El primero de ellos, OMA, es un elemento estructurante, que agrupa el ámbito de aplicación de un símbolo (una función). Contiene siempre un elemento de símbolo (OMS) seguido de los argumentos de este símbolo/función. Los argumentos pueden ser cualquiera de cuatro primeros elementos descritos y su número es siempre o uno o dos, dependiendo del número de argumentos que acepte esa función. Por ejemplo, la función seno acepta un argumento, mientras que la función raíz acepta dos, la base y el índice.

Es por esto que tenemos tres tipos de nodos en el árbol. Los nodos hoja que son las variables y los números, y los nodos rama, que pueden tener uno o dos hijos. Cada nodo debe obtener su propia evaluación. Los nodos hojas se evalúan por si solos, y los nodos rama, aplican su función a sus argumentos (sus hijos). Por lo tanto la evaluación total de la fórmula aquella de la raíz. Teniendo esto en cuenta, hemos creado una jerarquía de clases que reflejan esta situación. Toda fórmula implementa la interfaz *FormulaEvaluable*, que le permite evaluarse. Además, aquellas que son nodos rama implementan la interfaz *FormulaAnidable*, que les permite anidar hijos. El siguiente diagrama representa estas clases:

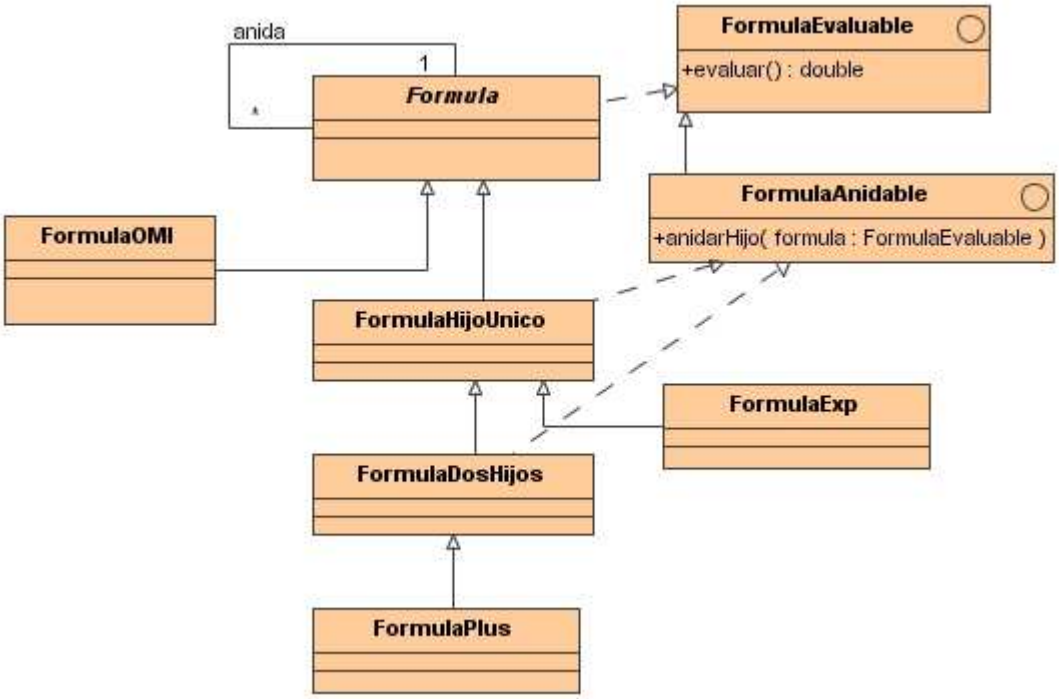


Figura 4.7: Diagrama de clases Formulas

Como ejemplo de fórmula hoja, tenemos *FormulaOMI*, que representa a un nodo de un número entero (OMI). Una fórmula de un sólo hijo es *FormulaExp*, que representa a la función exp(x) mientras que *FormulaPlus* representa la función suma (x+y), o sea, un nodo con dos hijos.

4.2.2 Comunicación Alumno Test

El primer requisito especificado en la fase de análisis de este componente era ampliar la funcionalidad del applet de Wiris. Como se vio en el apartado 2.3 y 2.5 (Planteamiento y Estado previo del problema, respectivamente), la herramienta de edición Wiris ofrece dos tipos de Applets, *EditorApplet*, que es una herramienta de edición, y *EditorTool*, que es una herramienta de presentación (aunque opcionalmente, también puede ser de edición, como se mostrará en el apartado 4.3.3.1). Ambas se configuran a través de parámetros HTML, y estos parámetros no son ampliables. El cometido de la comunicación entre el Alumno y el Test es hacer que la *EditorApplet* sea un applet de evaluación para SIETTE, y que *EditorTool* fuera la herramienta que usaríamos como presentación de la respuesta del alumno.

Por un lado, los applets de evaluación en SIETTE, extienden todos de una clase abstracta, SietteApplet, que provee de una interfaz que obliga a implementar los métodos *Evaluar*, *RespuestaPasiva* y *Resolver*. El primero de ellos, el más importante, es el que extrae del applet, la respuesta que el alumno ha introducido. El segundo, es el que llama el motor de test desde la página web a través de un entorno HTML mediante una comunicación Java-JavaScript para obtener la respuesta. Típicamente, este método delega en Evaluar. Resolver es el método que algunos applets pueden implementar para mostrar la respuesta correcta (si es que disponen de ella) cuando el alumno ya ha respondido y continua hacia la siguiente respuesta. En el caso de este proyecto, la respuesta correcta no depende del applet, si no de cómo haya definido el profesor la pregunta. La tarea de mostrar una solución correcta recae, pues, en el campo Ejemplo Válido de la pregunta de Respuesta Libre.

Esta interfaz plantea el problema de la doble herencia, que no tiene solución en Java (aunque si en otros lenguajes orientados a objetos). No podemos obtener toda la funcionalidad de *EditorApplet* de Wiris y además extender de *SietteApplet* para implementar la interfaz comentada. Por lo tanto, hubo que cambiar el diseño de *SietteApplet*, creando una interfaz independiente (*SietteAppletEvaluable*) que definiera estos tres métodos, y así poder extender del applet de Wiris.

Por otro lado, *EditorTool* es otro applet cuya configuración está cerrada, no pudiéndose añadir mas parámetros. Los requisitos obligaban a utilizar algún método de representación gráfica de la respuesta. Por desgracia, *EditorTool* no contaba con la posibilidad de introducir el contenido mediante parámetros. Este problema se solucionaba de forma sencilla, redefiniendo la función que inicializa el applet desde los parámetros y ampliando su funcionalidad.

Por lo tanto, el resultado de este diseño es el siguiente:

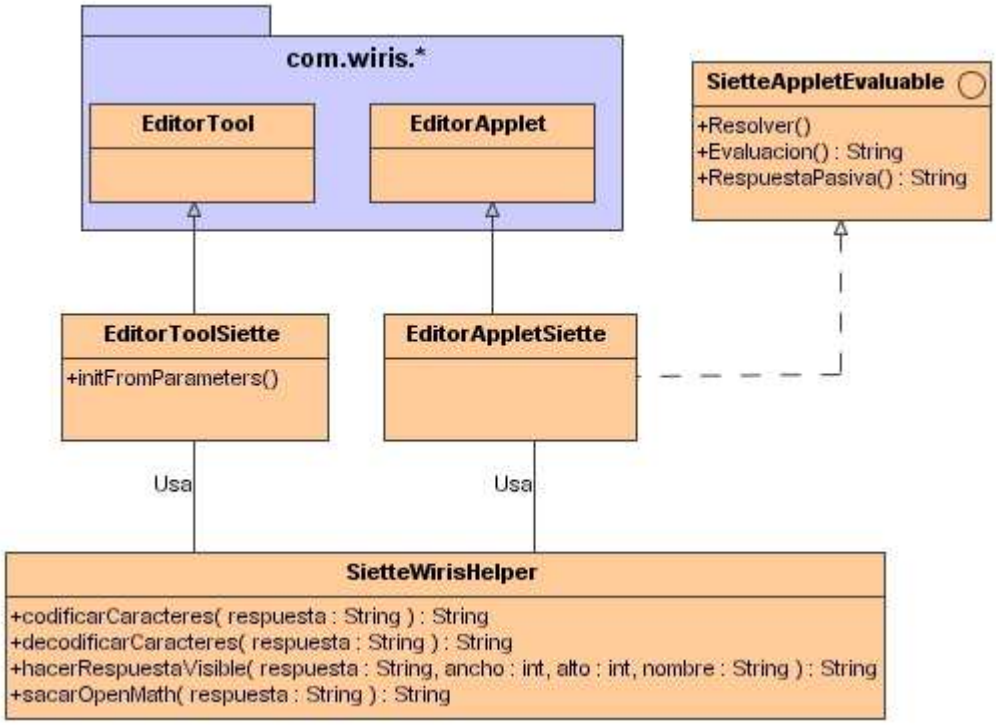


Figura 4.8: Diagrama de clases de Applets Visuales

La clase *SietteWirisHelper* es una clase que se utiliza en ambos lados de la comunicación. *EditorAppletSiette* llama al método *hacerRespuestaVisible* desde su método Evaluación, es decir, añade a la respuesta del alumno, dada por *EditorApplet* en código OpenMath, un "envoltorio" que es su representación en HTML.

Cuando la respuesta llega al servidor, el patrón *OpenMathPattern*, utiliza el método *sacarOpenMath* para obtener el contenido matemático del "envoltorio" de la respuesta. Los métodos *codificarCaracteres* y *decodificarCaracteres* son métodos para evitar el conflicto entre caracteres reservados de HTML que se usan en código OpenMath. Una explicación más detallada de este proceso se hace en el apartado de implementación de este componente, la sección 4.3.2.

4.2.3 Applets de edición

Los applets de edición son pequeñas aplicaciones que han de ser lanzadas como editores externos. Se establece entre los editores de tests y estas herramientas una comunicación Java-JavaScript que les permite acceder a los campos que han de editar. Todos los editores externos en Siette extienden de la clase *JApplet*, y deben implementar un par de métodos como mínimo, que son *procesarTexto()* y

generarCodigo(), ambos para la entrada y salida de datos respectivamente. Siette tenía implementado una clase abstracta, *EditorApplet*, que definía estos métodos. Otra vez, el problema de la doble herencia (sólo para el Applet de Respuestas) obligó a que ésta clase implementara una interfaz genérica llamada *SietteAppletEditable*, que contenía estos dos métodos, y así poder extender de otra clase. La siguiente figura muestra el esquema de este diseño:

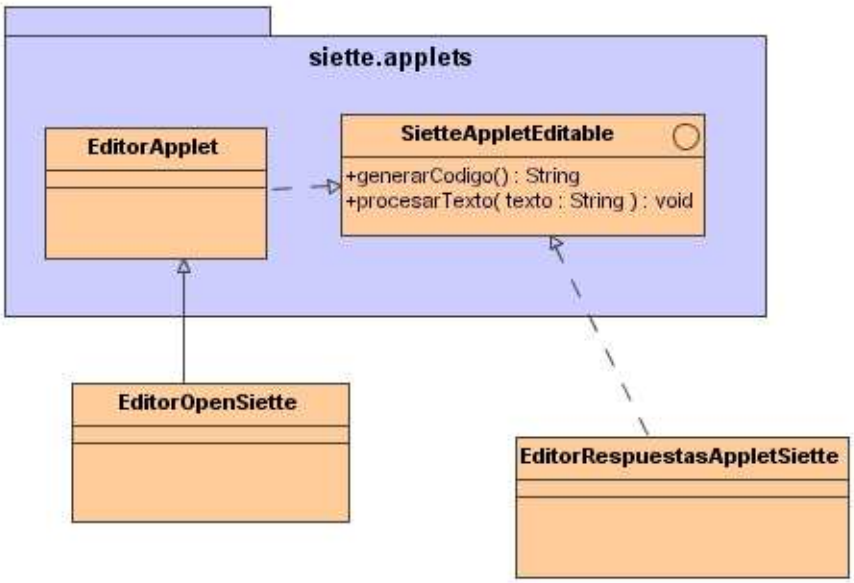


Figura 4.9: Diagrama de composición de Editores

4.2.3.1 Applet de Edición de Enunciados

El applet de edición de enunciados (EditorOpenSiette a partir de ahora) es una pequeña aplicación que sigue el conocido patrón de diseño de Modelo-Vista-Controlador. La vista es la interfaz de usuario que se muestra, el control son las acciones que se encuentran de los botones y comandos de esta interfaz y el modelo es la abstracción del documento que se está editando. Por encima de todo, está la aplicación, que inicializa estas tres partes, las enlaza y sirve como punto de entrada.

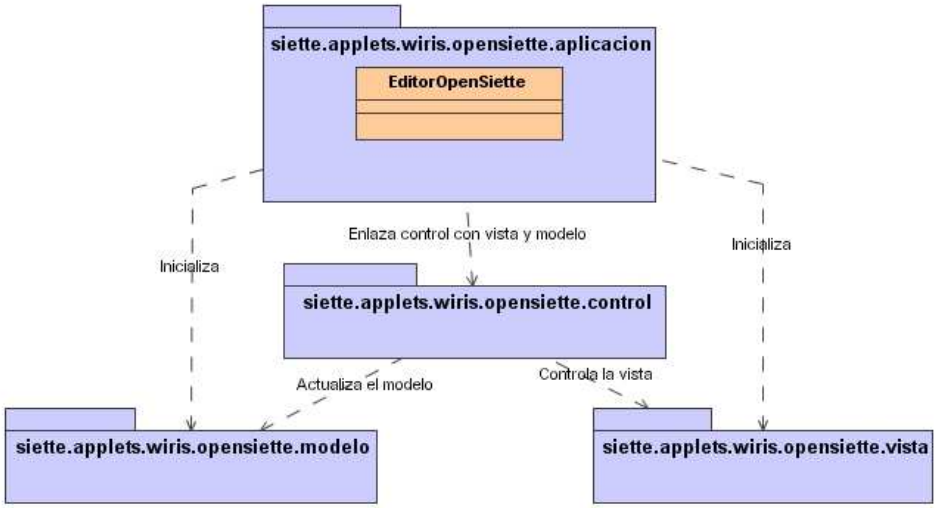


Figura 4.10: Diagrama de composición M-V-C

Desglosando esta estructura por partes tenemos:

El modelo:

El modelo es una abstracción de un documento HTML 4.0, implementado de manera sencilla. Está ideado como una secuencia de caracteres que tienen asociado un estilo. Como el modelo ha de ser coherente con la vista, y como se verá en la sección de la misma y en el manual de usuario, ésta permite asociar un estilo (un conjunto de formatos) no sólo a los caracteres del texto, sino también al hueco entre ellos, por lo que al empezar a escribir en ese punto se utiliza ese estilo. Es por esto que básicamente, el modelo del enunciado HTML se compone tres vectores, uno de caracteres, otro de los estilos asociados a los mismos y el tercero de estilos asociados a los puntos intermedios. Estos vectores han de ser dinámicos, ampliables en tiempo de ejecución.

Estos tres vectores se encuentran en la clase *EnunciadoHTML*. Además de estos tres atributos, esta clase cuenta, en un principio, con métodos para el manejo de estos caracteres: *insertar*, *borrar*, *cambiarEstilo*, *marcarEstilo* (éste último para asociar un estilo a un punto intermedio entre caracteres). Con respecto a la representación interna del modelo estas características eran suficientes. Pasemos ahora a describir otras cuestiones de diseño importantes relativas al modelo.

La entrada y salida de datos. Tal como se especificó en el análisis de requisitos, este editor debía aceptar un documento HTML cualquiera, editarlo visualmente y

devolverlo en formato HTML 4.0 (el más reciente). Para ello se utiliza un proceso de análisis gramatical y otro de escritura. Para el análisis se utilizó el mismo sistema que para el análisis de OpenMath en el evaluador. Dos clases, implementado el patrón Oyente, una que analiza el texto con un proceso LR descendiente, *AnalizadorXML*, y otra, *ManejadorXML* que recibe los tokens escaneados en el análisis y va construyendo la representación interna del documento a un objeto *EnunciadoHTML*. Para la escritura es la propia clase *EnunciadoHTML* la que a través del método *devolverHTML* crea el texto HTML, utilizando sólo los elementos permitidos por la versión 4.0 del estándar.

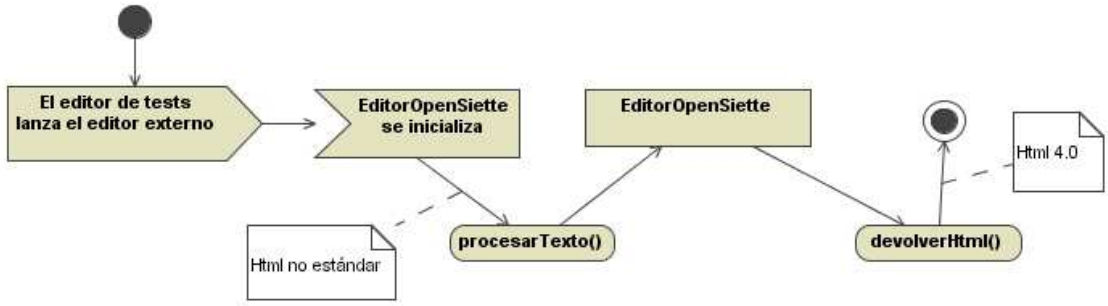


Figura 4.11: Diagrama de actividades de la entrada y salida

La sobrecarga de instanciaciones de objetos. En el diseño de editores hay que tener muy en cuenta el tamaño que puede llegar a alcanzar en memoria la representación del modelo. Si contamos que por cada carácter ha de almacenarse hasta tres objetos (como mínimo) en memoria, y que en un enunciado de SIETTE puede haber hasta 4000 caracteres, el uso de memoria es muy alto. Por otro lado tenemos que habría muchísimas repeticiones del mismo objeto. Cómo mucho existirán 26 caracteres diferentes, más unos 20 signos de puntuación y otros caracteres. Y en el caso más extremo, cada carácter tendrá un estilo propio y único, pero normalmente el estilo es común a grandes grupos de los mismos. En estos casos se usa el patrón de diseño *FlyWeight*, que permite reutilizar las mismas instancias para diferentes apariciones de un mismo objeto. Mediante un pool, o almacén de instancias, se crean, en este caso, como mucho unos 40 caracteres. Cuando se va creando el modelo del documento en el proceso de análisis, se utiliza la clase *FactoriaDeCaracteres* (que implementa el patrón *FactoryMethod*), la cual devuelve una instancia ya creada del carácter, o una nueva si era la primera aparición. Y por lo tanto, en los vectores internos de *EnunciadoHTML* sólo se guardan las referencias a estas instancias comunes. Para los estilos se sigue el mismo procedimiento. Hay que reseñar, que para que este patrón funcione, cuando se

demanda una instancia ya creada, está ha de ser unívoca, no puede haber repeticiones. Para los caracteres esto es fácil, ya que cada carácter es único, y Java permite fácilmente la comparación de este tipo. Pero para los estilos, cuyo diseño se comenta más adelante, hay que implementar la comparación de manera idónea, para evitar repeticiones y redundancias en este almacén de instancias. Cada estilo ha de tener una identificación única.

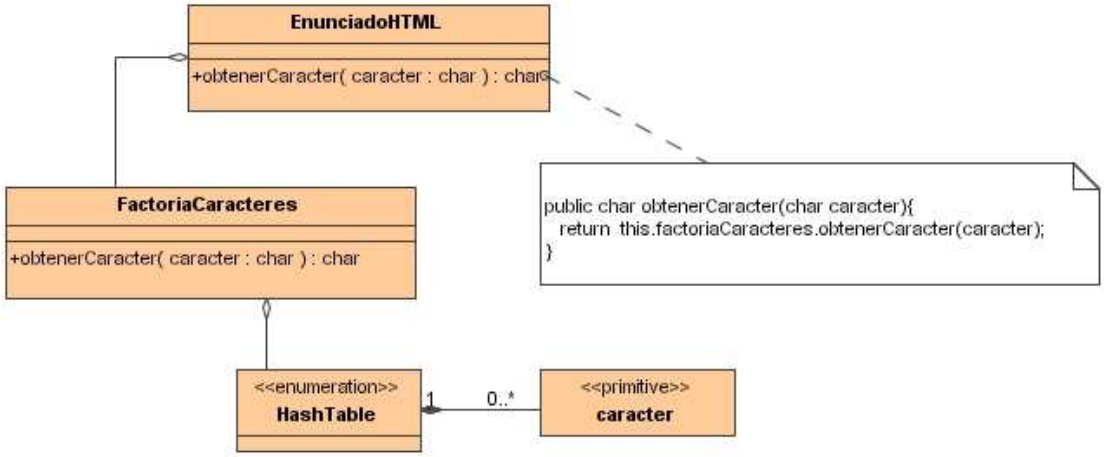


Figura 4.12: Diagrama de clases de FactoriaDeCaracteres

Hacer y deshacer. Hemos dicho que el modelo es una abstracción de lo que el usuario edita en la vista. Por ahora, con las funciones de insertar, borrar y modificar era suficiente para esta equivalencia. Pero, tal como se especificó en el análisis, el usuario puede hacer y deshacer cambios. Para ello, el modelo ha de mantener siempre la coherencia con lo que el usuario ve, haciendo y deshaciendo los cambios a la vez que la vista. En ésta, los paquetes que ofrece Java facilitan sobremanera este proceso. Pero para el modelo, hemos tenido que diseñar un sistema específico para esta tarea. El patrón de diseño *Command* se ajusta perfectamente a este propósito. Guardando en una pila las acciones que se han ido ejecutando, se permite deshacer y rehacer todo el historial de las mismas. Cada acción guardada ha de mantener ella misma el estado en el que se encontró el modelo y los mecanismos para recrear las veces que sea necesario el estado en el que lo deja. Para ello hemos diseñado la clase *Accion*, que declara los métodos *deshacer()* y *ejecutar()*. En la sección dedicada al Control, se verá como cada acción ejecutada en la vista por el usuario tiene su reflejo en una de estas Acciones. La clase auxiliar *ManagerAcciones* mantiene las pilas de acciones por deshacer y acciones por rehacer y es la encargada de recoger los eventos de deshacer y rehacer,

trasladándolo a las acciones de la cima de la pila correspondiente. Claro está que cuando una acción se deshace pasa a la pila de acciones por rehacer y viceversa. Una última clase, *AccionesAyuda*, un configurador, que incluye algunos métodos útiles del manejo del modelo para ejecutar las acciones completa el diseño de este sistema.

Los estilos. La representación de un estilo en el modelo incluye características como el formato (negrita, cursiva, subrayado), el color de la fuente o la alineación del texto. Por ello se creó el estilo como una estructura con estos valores. La existencia o no de estos valores determina el estilo del carácter. Algunos valores son complementarios, como cursiva o negrita, otros son suplementarios, como color rojo o azul, o alineación centrada o justificada. Para poder implementar correctamente el patrón *FlyWeight* comentado anteriormente e interiormente el patrón *FactoryMethod*, todo estilo ha de poder ser identificado unívocamente por una "clave" para poder buscarlo en el "diccionario" de instancias que se propone en este patrón. Para ello se genera una cadena de texto que representa el estilo, concatenando la representación textual de sus atributos, siempre en el mismo orden. Así conseguimos identificarlos sin ningún género de duda.

La vista:

Para el diseño de la vista se ha usado el paquete *javax.swing*, de la API de Java, usando la clase *JTextPane*, que es, básicamente, un panel de edición donde además de texto, se pueden incluir otros componentes visuales de Java, lo cual es ideal para incluir las fórmulas. Esta clase tiene varias características muy útiles. Por cada evento de edición que el usuario ejecuta, se lanza un aviso que ha de ser tratado por el mecanismo de control necesario por si, por ejemplo, se quiere reflejar este evento de edición en el modelo subyacente. Por otro lado se permite incluir texto mediante programación, y no sólo esperando a que el usuario lo edite. Este texto también puede tener formatos, pero no permite muchas posibilidades, aunque las suficientes para este proyecto. Entre los formatos, uno muy importante es, como se ha comentado, la posibilidad de que un texto en vez de caracterizarse por estar en negrita o en rojo, sea sustituido por un componente visual.

Como toda clase visual del paquete *javax.swing*, posee además de la vista, un modelo, una abstracción de un documento. Java permite varias posibilidades para este

modelo, como un *DocumentoRTF* o un *DocumentoHTML*. La razón de que no se haya usado este modelo y se haya implementado uno propio para el editor es que este modelo sólo permite abstraer documentos HTML hasta su versión 3.4, que no permitía la inclusión de applets, algo primordial para este proyecto. La posible ampliación de estas clases para adaptarlas a la versión 4.0 implicaba tales cambios que era más eficiente implementar una versión adaptada a las necesidades específicas del proyecto, que fuera sencilla de manejar y comprensible.

En la vista se incluyen métodos para la creación de las barra de herramientas, y para el acceso a algunas de sus propiedades.

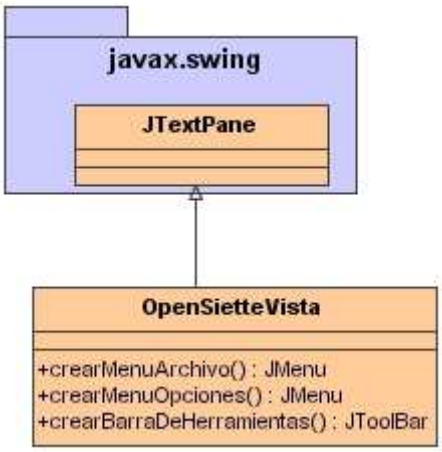


Figura 4.13: Diagrama de clases de la vista

El control. El control es un compendio de acciones y oyentes de eventos que actúan entre la vista y el modelo. Por un lado, las acciones, que son reflejo de las que el usuario puede utilizar en la vista. Las acciones tienen acceso al modelo y a la vista. Cada acción implementa un pequeño oyente de eventos que sólo le llegan los eventos de la acción a la que pertenecen. Por ejemplo, la clase *AccionCursiva* implementa el método *actionPerformed()* al cual sólo llegan eventos de edición con cursiva. Todas estas acciones extienden de la clase *AbstractAction* de Java, aunque algunas extienden de clases más específicas que proporciona Java para estos casos, como son las acciones de edición. Cuando se captura un evento de este tipo, la acción ha de realizar el correspondiente cambio en el modelo.

Los oyentes son unas clases que implementan métodos como el *actionPerformed*, pero para eventos de componentes específicos. La clase *OyenteDocumento* está preparada para actuar ante los eventos de inserción o borrado de texto. La clase *OyenteCursor* actualiza el estado de las variables que indican la posición del cursor o del texto seleccionado, para poder utilizarlas si se actualiza el modelo. *OyenteDeshacer* es una clase que junto a las *AccionDeshacer* y *AccionRehacer* controla las operaciones de deshacer y rehacer, coordina estas acciones con la vista y con el modelo. En este editor, a nivel de aplicación, existe un objeto, de la clase de Java *UndoManager* que es el que controla si se puede actualizar o no. Invocando el método *canUndo()* (o *canRedo()*) se comprueba si internamente, el editor puede deshacer la acción, y si es así, este manager actualiza los estados de las acciones (habilitándolas o deshabilitándolas de cara al usuario) e invoca los métodos necesarios para hacer lo mismo en el modelo. Por último, el *OyenteEventos* captura todos esos eventos que no han sido capturados por los métodos descritos anteriormente. Estos eventos son por ejemplo los producidos por acciones anónimas de la vista. En este editor, la única acción anónima es la que provoca el usuario al establecer que quiere que el enunciado disponga o no del código HTML necesario para representar el applet de respuesta del alumno (véase el manual de usuario).

Por último, como se ha comentado, la clase *EditorOpenSiette* es la que está por encima de todo. Sus funciones son enmarcar la vista dentro de una clase *JApplet*, llamar a los métodos que inicializan el modelo con los datos obtenidos a través de la comunicación Java-JavaScript que se mencionaron más arriba, enlazar el modelo y la vista con el control e inicializar correctamente las tres partes.

4.2.3.2 Applet de Respuestas

El applet de edición de respuestas para el profesor es una adaptación del applet que ofrece la API de Wiris para la edición de fórmulas, que es el mismo que se le ofrece al alumno para responder. Cómo se ha comentado, estos editores han de implementar la interfaz *SietteAppletEditable*, para poder extender de otras clases. Éste editor extiende de la clase de Wiris *EditorApplet*, y para poder acceder a los contenidos que se están editando simplemente invoca los métodos *setContent()* y *getContent()*.

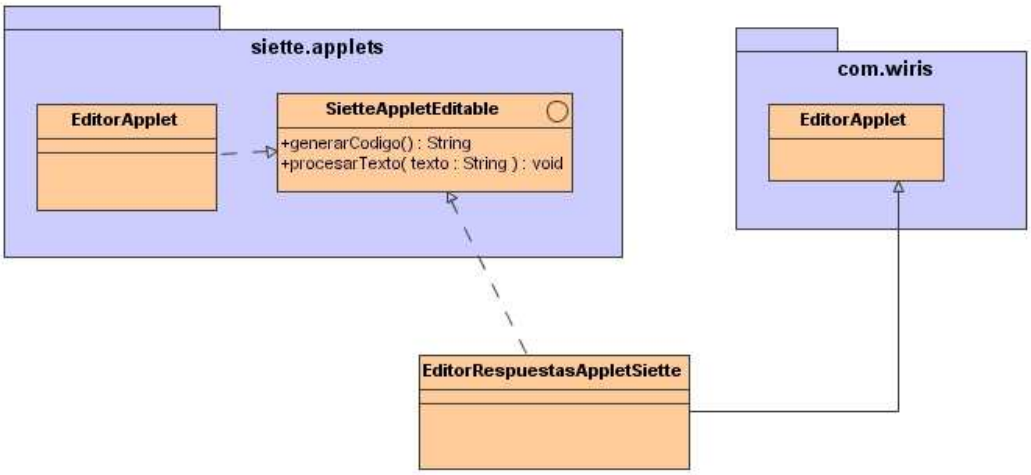


Figura 4.14: Diagrama de clases del editor de respuestas del profesor

4.3 Implementación

4.3.1 Evaluador

En este apartado se comentan los aspectos más relevantes de la codificación de las clases mostradas en el apartado 4.2.1

OpenMathPattern.java:

La función *pertenece* del patrón OpenMath implementa el siguiente código, que depende de la clase Comparador:

```
while (correccion == FALLO && i < patron.length) {
        if (respuesta != null && patron [i] != null) {
            try {
                Comparador c = new Comparador(patron [i],
                        normalizar(respuesta));
                if (c.evaluar()) {
                    correccion = i;
                }
            } catch (Exception e) {
                correccion = FALLO;
            }
        }
        i++;
    }
```

Esto es, por cada patrón, se crea el *Comparador* que compara la respuesta con el patrón. Si algún patrón coincide, se para de comprobar y se devuelve acierto. Si todos fallan, se devuelve fallo.

Para incluir el patrón dentro de las posibilidades ya existentes, hubo que modificar una parte del código y actualizar la Base de Datos. Estas son las actuaciones que se hicieron:

```
public void crearPatron() {
    if (aPatrones == null)
        aPatrones = obtenerListaPatrones();

    switch (idtipoPatron) {
        case SIETTE.PATRON_OPENMATH :
            patron = new OpenMathPattern(aPatrones, bModoMayusculas,
                    bModoAcentos, bModoSignos, bModoBlancos);
            break;
        case SIETTE.PATRON_JAVA :
            patron = new JavaPattern(aPatrones, bModoMayusculas,
bModoAcentos,
                    bModoSignos, bModoBlancos);
            break;
        case SIETTE.PATRON_EXP_REGULAR :
            patron = new ExpresionRegular(aPatrones, bModoMayusculas,
                    bModoAcentos, bModoSignos, bModoBlancos);
            break;
        case SIETTE.PATRON_CORRESPONDENCIA :
        default :
            patron = new Correspondencia(aPatrones, bModoMayusculas,
                    bModoAcentos, bModoSignos, bModoBlancos);
    }
}
```

Y en la scripts de creación de la Base de Datos se añadió el siguiente código:

```
INSERT INTO TIPOS_PATRONES ( IDTERMINO,NOMBRE,IDPATRON )
VALUES ( 542,'Patron OpenMath',4 ) ;
```

Comparador.java:

Como ya se ha comentado, por cada comparación se intentan hacer 10 evaluaciones de ambas fórmulas, comparando si en todos los casos los resultados numéricos son iguales (con un margen de error). Si en los 10 casos el resultado es satisfactorio, la respuesta es correcta. Si en alguno no lo es, la respuesta es incorrecta. Si en alguno de los casos no se pudiera hacer la evaluación, debido a que los valores aleatorios han coincidido que no pertenecen al dominio de la fórmula matemática dada, la evaluación no se cuenta. Si las 10 evaluaciones fallan, se pasa a comprobar la igualdad sintáctica. Éste también es el caso de que algunas de las fórmulas no sea válida por alguna razón. El siguiente gráfico muestra la secuencia de acciones de la función evaluar.

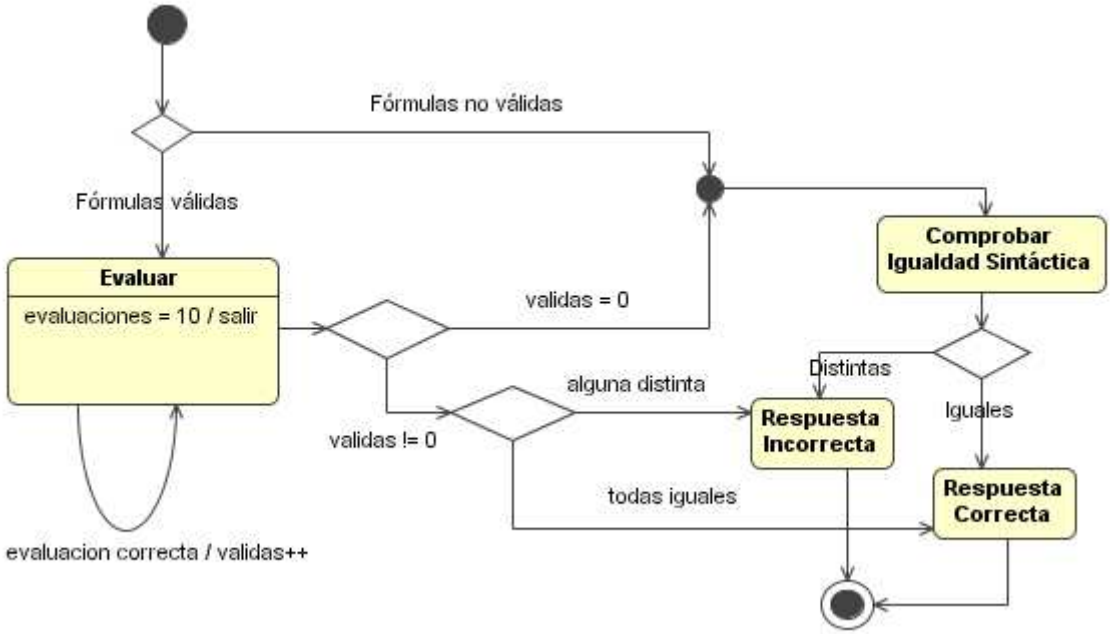


Figura 4.15: Diagrama de actividades del evaluador

Manejador.java y *XMLParser.java*

Estas dos clases implementan un analizador de lenguajes descendente tipo LR(1). A través de una máquina de estados y una pila, se va leyendo el texto de entrada, y se va analizando el texto XML. Cuando se reduce una secuencia de caracteres a un token concreto, el analizador lanza un evento, que recoge el manejador. Estos eventos indican el comienzo de un elemento (<nombre [atri="valor"]*>), el fin de un elemento (</nombre>), la aparición de texto o el principio y fin del documento.

Cuando se lanza un evento, al manejador se le pasa también el elemento que lo ha causado, si es un evento de comienzo o fin de elemento, así como sus atributos. Así, el manejador es capaz de crear la estructura arbórea de la fórmula matemática que el texto XML analizado representa. El analizado del texto se hace de izquierda a derecha, y OpenMath está codificado para que esta lectura haga un recorrido primero en profundidad del árbol, por lo tanto sólo necesitamos mantener en memoria en cada momento el nodo con el que se está trabajando en ese momento.

Uno de los requisitos del evaluador era que la comprobación de la corrección de las fórmulas no fuera una carga muy pesada para SIETTE, pues hay que hacer una por cada patrón, por cada respuesta, y por cada alumno (recordemos que SIETTE es un sistema para hacer test simultáneos). Por eso se desestimó el uso de API's de manejo de

XML estándares, como JDOM, o SAX, que aunque ofrezcan más funcionalidad, aportarían una carga extra muy grande comparada con la simplicidad de estas dos clases.

4.3.2 Comunicación Alumno Test

Como se vio en el diseño de este componente, la clase *SietteWirisHelper* juega un papel muy importante en el proceso, porque es la que envuelve la respuesta OpenMath del alumno en su propia representación HTML. El problema principal de utilizar HTML como representación es que el código OpenMath, al estar expresado en XML, contiene caracteres reservados propios de HTML también, como son los ángulos de apertura y cierre ,'<' y '>'. También contiene caracteres que pueden malinterpretarse cuando van incluidos dentro un tag de parámetro (PARAM) como son los espacios, las comillas dobles o los saltos de línea. La idea de incluir el código OpenMath dentro de su representación se resuelve en incluirlo como parámetro del applet *EditorToolOpenSiette*. El resultado final de hacer una respuesta OpenMath visible en HTML es el siguiente:

```
<applet      mayscript
       name='formula'
       code='siette.applets.wiris.EditorToolSiette'
       codebase='/siette'
       archive='lib/wiris.jar,lib/sietteapplet.jar'
       width='206'
       height='64'
       align='center'>
      <param name='domainURL'
value='/siette.htdocs/wiris/domains/globalAlumno.xml'>
      <param name='fontsDir' value='fonts/'>
      <param name='editableAll' value='false'/>
      <param name='content' value='AQUI IRIA EL CONTENIDO'/>
</applet>
```

El hecho de que el contenido vaya entre comillas por ser el valor del atributo "*value*" de la etiqueta "*param*" hace que la inclusión de estos caracteres pueda confundir a un navegador y lo muestren de manera equivocada. Por eso, la clase *SietteWirisHelper* codifica estos caracteres especiales antes de incluirlos en la respuesta, convirtiendo el código OpenMath en texto plano. Luego el servidor los decodifica y los devuelve a su estado original.

Estos caracteres se han codificado usando algo parecido a las entidades de HTML, esto es, usando '**&**' seguido de un código de dos o tres letras y finalizado por un '**;**'. Es importante notar que no se pueden usar las entidades estándar de HTML como '**<**' o '**>**' porque los navegadores las interpretarían y las convertirían directamente en '**<**' o '**>**', confundiendo así al decodificador dentro del applet *EditorToolSiette*.

Esta clase recoge el contenido que tiene que mostrar a través del siguiente código:

```
protected void initFromParameters(){
    super.initFromParameters();
    String value = getParameter("content");
    if(value != null)
    {
      SietteWirisHelper helper = new SietteWirisHelper();
      this.setContent(helper.decodificarCaracteres(value));
    }
}
```

Por otra parte, el applet EditorAppletSiette, que es el que genera la respuesta del alumno más su representación, actúa de la siguiente manera:

```
public String Evaluacion() {
    String st =  getContent();
    SietteWirisHelper helper = new SietteWirisHelper();
    int width = this.formulaArea.getImage().getWidth();
    int height = this.formulaArea.getImage().getHeight();
    return helper.hacerRespuestaVisible(st,width,height,"respuesta");
}
```

Y en el lado del servidor, es la clase OpenMathPattern la que se encarga de extraer la respuesta del alumno:

```
public int pertenece(String respuestaAlumno) {
    SietteWirisHelper helper = new SietteWirisHelper();
    String respuesta = helper.sacarOpenmath(respuestaAlumno);
    ...
```

4.3.3 Applets de Edición

La implementación de los applets de edición comprende la mayoría del código escrito. En la sección dedicada al diseño de ambos se explican extensamente el uso que

se le da a las clases. En ésta sección se pretende clarificar algunos aspectos que no hayan quedado claros o no se hayan mencionado, o que por su complejidad, necesiten del apoyo del código para su mejor comprensión.

4.3.3.1 Applet de Edición de Enunciados

En la implementación del modelo del editor podemos destacar varios aspectos importantes, de los que hemos hablado a grandes rasgos. Hicimos notar que en el proceso de entrada y salida se actualiza el código HTML a la versión 4.0 del estándar. Esto se hace a través del uso de propiedades CSS 2.0 en el atributo style de elementos span, que simplemente sirven para agrupar texto con una mismas propiedades. De la creación del código HTML necesario que envuelve la cadena de texto con estos atributos y este elemento span se encarga la clase *Estilo*, a través del método *crearHtml*:

```
public String crearHtml(String cadena) {
            String res = "";
            List formato = this.getFormato();
            String estilos = "";
            for (int i = 0; i < formato.size(); i++) {
                  String f = (String) formato.get(i);
                  if (f.equals(Estilo.NEGRITA)) {
                        estilos += "font-weight:bold;";
                  }
                  if (f.equals(Estilo.CURSIVA)) {
                        estilos += "font-style:italic;";
                  }
                  if (f.equals(Estilo.SUBRAYADO)) {
                        estilos += "text-decoration:underline;";
                  }
            }
            List alineacion = this.getAlineacion();
            for (int i = 0; i < alineacion.size(); i++) {
                  String f = (String) alineacion.get(i);
                  if (f.equals(Estilo.IZQUIERDA)) {
                        estilos += "text-align:left;";
                  } else if (f.equals(Estilo.DERECHA)) {
                        estilos += "text-align:right;";
                  } else if (f.equals(Estilo.CENTRADO)) {
                        estilos += "text-align:center;";
                  } else if (f.equals(Estilo.JUSTIFICADO)) {
                        estilos += "text-align:justified;";
                  }
            }
            List color = this.getColor();
            for (int i = 0; i < color.size(); i++) {
                  String f = (String) color.get(i);
                  if (f.equals(Estilo.AZUL)) {
                        estilos += "color:blue;";
                  } else if (f.equals(Estilo.ROJO)) {
                        estilos += "color:red;";
                  } else if (f.equals(Estilo.NEGRO)) {
```

```
                        estilos += "color:black;";
                  }
            }
            if (!estilos.equals("")) {
                  res = "<span \n style=\"" + estilos + "\">" + cadena
+ "</span>";
            } else {
                  res = cadena;
            }
            return res;
      }
```

El estilo, cuyos atributos se guardaban en los vectores color, formato y alineación, va procesando estos valores y en función de su aparición o no, incluye en el atributo style la concatenación de las propiedades CSS 2.0. Haciendo esto para todos los estilos del modelo y sus textos asociados, se procesa el documento entero.

Se comentó que la sobrecarga de instancias en memoria se evitaba utilizando el patrón *FlyWeight* junto al patrón *FactoryMethod*, creando un pool, o almacén, de objetos que identificados unívocamente podían reutilizarse. Para el caso de los caracteres, éste código es el que posibilita tal comportamiento:

```
public class FactoriaCaracteres {
      private Hashtable poolCaracteres;

      public FactoriaCaracteres(){
            poolCaracteres = new Hashtable();
      }

      public Character obtenerCaracter(char c){
            Character caracter = new Character(c);
            Character carResp =
(Character)poolCaracteres.get(caracter);
            if(carResp == null){
                  carResp = new Character(c);
                  poolCaracteres.put(carResp, carResp);
            }
            return carResp;
      }

}
```

El método *obtenerCaracter* es el que devuelve el objeto a reutilizar, a través de su identificador único, en este caso, el propio carácter.

Para la implementación del patrón *Command*, que era el que permitía la abstracción de acciones de usuario en objetos que se pueden almacenar y que aisladamente, ellos sólo puedan recrear el estado anterior del documento o volverlo a

modificar. Se implementó la clase abstracta Acción, que definía los métodos deshacer y ejecutar. En el siguiente código la implementación in-line de esta clase, en este caso cuando el usuario procede a formatear algún texto en Negrita, dentro del método *actionPerformed*. Las variables *doc* y *estadoFinal* apuntan, respectivamente, al modelo (*EnunciadoHtml*) y una cadena de texto que es el estado en el que estaba el modelo antes de la acción. Los métodos deshacer y ejecutar se valen de estas variables para poder implementar correctamente en el futuro su cometido.

```
Accion accion = new Accion(){
            public void ejecutar(){
                  doc.resetear();
                  doc.inicializar("<html>"+estadoFinal+"</html>");
            }
            public void deshacer(){
                  doc.resetear();
                  doc.inicializar("<html>"+estadoInicial+"</html>");
            }
        };

        this.doc.getManagerAcciones().insertarAccion(accion);
```

La implementación de la vista es bastante corta, con respecto al resto de la aplicación. Destacamos, por ejemplo, el uso de las acciones para crear los botones de las barras de herramientas, que ahorran tener que definir muchos aspectos de los mismos pues los heredan de la acción a la que representan.

```
barra.add(boton);
boton = new JButton();
boton.setAction(aReh);
boton.setBackground(new Color(90, 117, 148));
boton.setText(null);
boton.setBorder(null);
boton.setIcon(new
ImageIcon(this.getClass().getResource("/siette/applets/wiris/opensiett
e/recursos/redo.gif")));
```

En este pequeño trozo de código se observa que a los botones sólo se le asignan su color y su icono, el resto de atributos, como el comando de la acción que representa, o el controlador asociado lo toman de la acción (*aReh*, en este caso, la acción de Rehacer). Otro aspecto importante que antes no se había mencionado es la existencia del paquete de recursos, que contiene las imágenes y los archivos de configuración del editor.

Veamos a continuación algunos aspectos del controlador que merecen la pena ser destacados. Las acciones son todas ampliaciones de acciones específicas que la API de Java ofrece para el manejo de editores de texto. En este caso, la *AccionCursiva* hereda de una *ItalicAction*. La clase envolvente *StyledEditorKit* es un compendio de clases útiles para el manejo de los editores:

```
public class AccionCursiva extends StyledEditorKit.ItalicAction {

private EnunciadoHtml doc;


public void actionPerformed(ActionEvent e) {
    super.actionPerformed(e);
    if (doc.modeloEditable) {
      final String estadoInicial = doc.devolverHtml();
        AccionesAyuda ayuda = new AccionesAyuda(this.doc);
        ayuda.cambiarSeleccion(Estilo.FORMATO, Estilo.CURSIVA);
        final String estadoFinal = doc.devolverHtml();

        Accion accion = new Accion(){
            public void ejecutar(){
                  doc.resetear();
                  doc.inicializar("<html>"+estadoFinal+"</html>");
            }
            public void deshacer(){
                  doc.resetear();
                  doc.inicializar("<html>"+estadoInicial+"</html>");
            }
        };

        this.doc.getManagerAcciones().insertarAccion(accion);
    }
}

/**
 * @param doc the doc to set
 */
public void setDoc(EnunciadoHtml doc) {
    this.doc = doc;
}
}
```

La variable *modeloEditable* del modelo es muy importante, pues indica si este es receptivo o no a las acciones disparadas por el usuario. Esto es muy relevante al inicio de la aplicación, donde insertamos texto manualmente, lo cual dispara estas acciones, y no queremos que estos eventos repercutan en el modelo, pues todavía no ha empezado la edición por parte del usuario.

Vemos esto también en el *OyenteDocumento*. Su comportamiento ante un evento de inserción de texto es, si el modelo es editable, utilizar los métodos que ofrecía para insertar el texto, que ha obtenido del evento.

```
public void insertUpdate(DocumentEvent e) {
    int tam = e.getLength();
    int inicio = e.getOffset();
    try {
      final String estadoInicial = doc.devolverHtml();
        if (doc.modeloEditable) {
            this.doc.insertar(inicio,   e.getDocument().getText(inicio,
tam));
```

La variable e representa el evento, y contiene el documento asociado a la vista, de donde se puede obtener el texto insertado por el usuario.

Por último, vamos a mostrar la secuencia de acciones del inicio de la aplicación, en el método *init* de la clase *EditorOpenSiette*, que es el primero que llama el navegador cuando se carga el applet:

```
public void init() {
            vista = new OpenSietteVista();
            modelo = new EnunciadoHtml();

            this.inicializarAcciones();

            oyenteDoc = new OyenteDocumento();
            oyenteDoc.setDoc(this.modelo);
            oyenteCursor = new OyenteCursor();
            oyenteDeshacer = new OyenteDeshacer();
            oyenteDeshacer.setModelo(modelo);

            managerDeshacer = new UndoManager();
            managerDeshacer.setLimit(limiteDeshacer);
            oyenteDeshacer.setADes(this.aDes);
            oyenteDeshacer.setAReh(this.aReh);
            oyenteDeshacer.setManager(this.managerDeshacer);

            this.aDes.setAReh(this.aReh);
            this.aDes.setManager(this.managerDeshacer);

            this.aReh.setADes(this.aDes);
            this.aReh.setManager(this.managerDeshacer);

            vista.addCaretListener(oyenteCursor);
            vista.getDoc().addDocumentListener(oyenteDoc);
            vista.getDoc().addUndoableEditListener(oyenteDeshacer);
            oyenteEventos = new OyenteEventos();
            oyenteEventos.setDoc(this.modelo);
            vista.setOyeEventos(oyenteEventos);
```

```
            vista.setACFAzul(this.aCFAzul);
            vista.setACFNegro(this.aCFNegro);
            vista.setACFRojo(this.aCFRojo);
            vista.setACur(this.aCur);
            vista.setANeg(this.aNeg);
            vista.setASub(this.aSub);
            vista.setADes(this.aDes);
            vista.setAReh(this.aReh);
            vista.setAACen(this.aACen);
            vista.setAADer(this.aADer);
            vista.setAAIzq(this.aAIzq);
            vista.setAAJus(this.aAJus);
            vista.setAFor(this.aFor);
            vista.setASal(this.aSal);
            vista.setACer(this.aCer);

            JMenu menuArchivo = vista.crearMenuArchivo();
            JMenu menuOpciones = vista.crearMenuOpciones();
            JToolBar barraHerr = vista.crearBarraHerramientas();
            JScrollPane panel = new JScrollPane(vista);
            this.getContentPane().add(panel, BorderLayout.CENTER);
            this.getContentPane().add(barraHerr, BorderLayout.NORTH);
            vista.setSize(new Dimension(400, 400));
            JMenuBar mb = new JMenuBar();
            mb.add(menuArchivo);
            mb.add(menuOpciones);
            setJMenuBar(mb);

            setSize(new Dimension(400, 400));
            setVisible(true);
            try {
                  super.init();
            } catch (Throwable e) {
                  System.out.println("Excepcion en super.init():
"+e.getMessage());
            }
      }
```

La secuencia es: Crear el modelo y la vista, inicializar las acciones con estos. Luego, inicializar los controladores, enlazarlos con el modelo y la vista, coordinar el sistema de deshacer y rehacer, asociar a la vista las acciones para los botones, crear la vista y mostrarla. Finalmente, el método *init()* de la clase padre (*EditorApplet*) ejecutará la comunicación Java-JavaScript y llamará al método *procesarTexto()* con el enunciado obtenido de los campos del editor de test.

4.3.3.2 Applet de Edición de Respuestas

El applet de edición de respuestas simplemente hereda de *EditorApplet* (la clase de la API de Wiris, no la de SIETTE) e implementa los métodos para la entrada y salida de datos.

```
public class EditorRespuestasAppletSiette extends EditorApplet
implements SietteAppletEditable {

    public void init(){
        super.init();
        JSObject jsobject = JSObject.getWindow(this);
        String texto = (String) jsobject.eval("recogerDatos()");

        procesarTexto(texto);
    }

    public String generarCodigo() {
        String res = this.getContent();
        SietteWirisHelper h = new SietteWirisHelper();
        return h.sustituir(res, "  ", " ");
    }

    public String generarTexto(){
        return this.generarCodigo();
    }

    public void procesarTexto(String texto) {
        this.setContent(texto);
    }

}
```

Aquí observamos como en el método init, se ejecuta la comunicación Java-JavaScript. A través de la clase *JSObject*, de unas librerías de *Netscape*, se accede al método *recogerDatos()* de la página HTML envolvente del editor, que devolverá el texto a editar.

Se observa aquí, que al devolver el código editado por el usuario, se eliminan los espacios repetidos, que no aportan nada y pueden influir en un mal funcionamiento de SIETTE si la respuesta ocupa más de lo que se pueda almacenar en la Base de Datos.

4.4 Integración y pruebas

En esta sección se van a comentar algunos aspectos sobre el proceso software del proyecto que no se han nombrado antes y que son interesantes para comprender el desarrollo del mismo.

Por un lado, el espacio de trabajo ha sido siempre el entorno de desarrollo de Eclipse [43], de IBM, una herramienta completa y gratuita adecuada para todo tipo de proyectos con Java en general, aunque dado que es muy ampliable, se pueden utilizar

para otros lenguajes. En el caso de este proyecto, se amplió con plug-ins para la edición de ficheros HTML, JSP, XML, CSS o librerías JavaScript.

Dado que SIETTE es un proyecto que está en continuo desarrollo y mejora, se utilizaba un repositorio, un sistema de control de versiones, concretamente la herramienta CVS, para poder tener siempre los últimos cambios actualizados. Este proyecto se ha enmarcado dentro de este desarrollo y mejora, y por lo tanto, se ha usado intensivamente esta herramienta, así como otras herramientas de Gestión de la Configuración para coordinar las tareas del mismo con las demás de SIETTE.

Para ir probando el sistema a medida que iba creciendo, se han usado las herramientas de compilación, empaquetación y despliegue que existen en el entorno de desarrollo de SIETTE, utilizando siempre una Base de Datos de pruebas local a la máquina del alumno. Tanto las pruebas de interfaz, como las de usabilidad o de unidad se han desarrollado en este entorno, que es idéntico al que habrá en el sitio público de SIETTE cuando se considere oportuno desplegar los nuevos cambios en el mismo.

Para la depuración, tanto local como remota, se ha utilizado las herramientas que ofrece Eclipse junto al protocolo de comunicaciones SSH para cuando era remota (y en concreto la herramienta Putty).

Existen varios aspectos no comentados en este capítulo que pertenecen a la implementación, pero que no han entrado en el proceso software como tal. Son aspectos referentes a la integración del proyecto dentro de SIETTE.

Por ejemplo, para el correcto lanzamiento de los editores se hubo de crear una página JSP que incluyera el código HTML, JSP y JavaScript necesario para ello.

Igualmente, en la Base de Datos hubo que incluir en la tabla de editores externos el nuevo editor que se había creado, al igual que se hizo para la inclusión del nuevo patrón de corrección.

Otro aspecto importante son los archivos de configuración de los applets de Wiris. Hemos comentado que tanto el alumno como el profesor al editar las respuestas y

patrones respectivamente, utilizan un dominio restringido de fórmulas y operadores matemáticos. Sin embargo, para los enunciados, el dominio es completo, pudiendo mostrar cualquier tipo de fórmula, sin restricciones. Estos dos dominios son ficheros XML, *globalAlumo.xml* y *globalProfesor.xml* que hay que incluir en el directorio del servidor de aplicaciones donde se despliega SIETTE, concretamente en */webapps/siette.htdocs/wiris/domains/*.

Por último, comentar que para el correcto funcionamiento de las herramientas de compilación, empaquetación y despliegue de SIETTE, hubo de modificar el fichero de configuración *build.xm*l del que se sirve la herramienta ant para incluir los comandos necesarios para incluir el código dentro de la aplicación *siette.war*.

Capítulo 5
Caso de Uso: Una asignatura de Cálculo en SIETTE

5 Caso de Uso: Una asignatura de Cálculo en SIETTE

El objetivo primordial de este proyecto era ampliar las funcionalidades de SIETTE para poder crear asignaturas que pudieran contener ítems cuyas respuestas sean, en general, fórmulas matemáticas. Además, se pretendía lograr que para ello, tanto el alumno como el profesor tuvieran las máximas facilidades posibles, y no tener que preocuparse más que de en el caso del profesor redactar enunciados claros y respuestas correctas y en el del alumno de conocer la respuesta adecuada. Ninguno de ellos debería tener en cuenta la representación interna de estas fórmulas.

Además, el problema de tener un amplio abanico de respuestas equivalentes para un mismo problema se eliminaba, como se comentaba tanto en el capítulo 2 como en la implementación (Capítulo 4), gracias a la comprobación en base a la igualdad semántica.

Enumerando estas ampliaciones, contamos que, por un lado, se han creado para el profesor una serie de herramientas que le facilitan la tarea de creación de ítems de SIETTE. El uso del editor de enunciados OpenSiette le permite crear enunciados altamente complejos, con fórmulas matemáticas incluidas a través de una sencilla y funcional interfaz. Y para definir la respuesta o respuestas correctas a ese enunciado, el uso de este mismo editor en su versión para las respuestas le ahorra la necesidad de conocer el estándar OpenMath en el que las respuestas están representadas internamente. Estas respuestas están compuestas por un conjunto de funciones y operadores matemáticos, que aunque restringido, da posibilidad a infinidad de combinaciones, que cómo se verá en éste capítulo, es más que suficiente para casi cualquier cuestión imaginable. Hay que destacar que esta restricción no está presente en las fórmulas que puedan aparecer en el resultado.

Por otro lado, el alumno, a la hora de realizar a un test de contenidos creados con las herramientas que se han mencionado, no ha de temer enfrentarse a, además de la dificultad propia del test, a una complicada manera de establecer su respuesta. El uso del editor de fórmulas incluido en los enunciados para responder presenta la misma sencillez que el resto de las herramientas, con el mismo domino restringido de

operadores y funciones que hace de la edición de la respuesta una tarea fácil una vez se sabe la respuesta.

En este capítulo, vamos a mostrar paso a paso, el uso de estas herramientas en un caso de uso real de la creación de una asignatura de Cálculo, con contenidos de nivel de Bachillerato y Universitario. Para ello hemos tomado como referencia uno de los libros recomendados en la asignatura de Cálculo para la Computación de la ETSI de Ingeniería Informática de Málaga, "Cálculo Diferencial e Integral", de Frank Ayres Jr[2]. Es un compendio de pequeñas introducciones teóricas a los contenidos y un abundante espectro de ejercicios resueltos de una gran variedad de temas, desde Funciones, Límites o Derivadas, a Integrales Triples, Diferenciales o Cálculo de Volúmenes.

Dividiremos entonces este capítulo en dos secciones. El uso de estas herramientas desde el punto de vista del profesor, y desde el punto de vista del alumno.

5.1 La visión del profesor

Inicialmente el profesor ha de tener una cuenta dentro de la herramienta SIETTE. La creación y opciones de modificación de esta cuenta no son del ámbito de este proyecto, y simplemente se comenta por que es el requisito necesario.

Comentamos que actualmente en SIETTE existen dos herramientas de autor para editar los tests y los contenidos, TEDI y AthoS, siendo esta última la de más reciente implementación. Para este caso de uso usaremos la segunda, por ser con la que el autor ha tenido más experiencia, pero como se explicó en la sección 4.4, las herramientas desarrolladas se han integrado tanto en TEDI como en AthoS.

Dentro de su cuenta, puede tener ya creada una asignatura, o puede crear una nueva. En ambos casos, en las opciones de configuración generales de esta asignatura, ha de especificarse que el editor de la asignatura sea OpenSiette. Otros aspectos como el número de niveles de conocimiento, el nombre de la asignatura o la activación también son modificables. De ellos, la activación es otro que nos interesa, pues hará públicos la asignatura para que cualquier alumno pueda acceder a los tests que creemos.

Dentro de esta asignatura se puede crear un árbol de temas e ítems, organizado como si fuera los contenidos de un libro, por capítulos. El nivel de anidación es ilimitado, pero en este caso de uso hemos intentado abarcar los máximos temas posibles, y que se noten a simple vista, creando casi todos los temas en el primer nivel de anidación. Esta asignatura que se ha creado expresamente para probar este proyecto se incluirá en el sitio público de SIETTE, y servirá de referencia para aquellos que quieran crear asignaturas con contenido matemático. Por ello, la localización a simple vista de un tema específico era primordial.

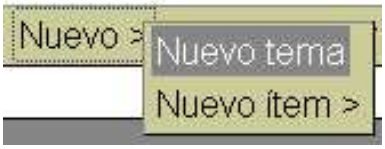


Figura 5.1: Botón de nuevo tema

Prácticamente se ha creado un tema por cada uno del libro antes referenciado. Para crear un nuevo tema simplemente hay que posicionarse en el tema padre (en nuestro caso la asignatura) y crear un nuevo tema a través de la barra de herramientas de que ofrece AthoS. Para este caso de uso hemos creado un total de 40 temas principales. Los enumeramos a continuación:

- Funciones
- Límites
- La derivada
- Derivación implícita
- Tangentes y normales
- Valores máximos y mínimos
- Aplicaciones de máximos y mínimos
- Derivación
- Representación paramétrica de curvas
- El teorema de la media

- Límites indeterminados
- Diferenciales
- Aplicaciones de las integrales indefinidas
- La integral definida
- Áreas planas de por integración
- Volúmenes sólidos de revolución
- Volúmenes de sólidos con secciones conocidas
- Longitud de arco
- Área de una superficie de revolución
- Integrales impropias
- Sucesiones infinitas y series
- Criterios para la convergencia y divergencia de series positivas
- Series con términos negativos
- Cálculo con series
- Series de potencias
- Desarrollos de funciones en series de potencias
- Series de McLaurin y Taylor con restos
- Cálculo con series de potencias
- Derivadas parciales
- Diferenciales totales y derivadas totales
- Funciones implícitas
- Vectores en el espacio
- Curvas y superficies en el espacio
- Derivadas direccionales
- Integrales dobles iteradas
- Volumen bajo una superficie por integración doble
- Área de una superficie por integración doble
- Integrales triples
- Ecuaciones diferenciales
- Integración

Todos estos temas contienen entre uno y cuatro ítems que ejemplifican su uso, salvo el último, que contiene una veintena de integrales.

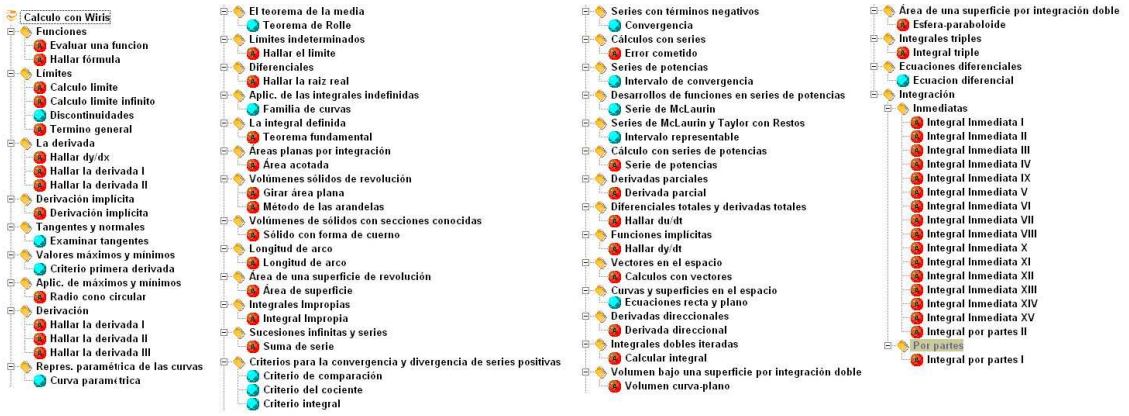


Figura 5.2: Árbol de ítems de "Cálculo con Wiris"

Una vez creados todos los temas, nos vamos a centrar en la creación de un ítem de respuesta corta, ya que para éstos es para los que hemos implementado el nuevo patrón de corrección OpenMath, y que son los que permiten al alumno responder libremente. Más adelante se explicará que también se han creado ítems, que dada la naturaleza de lo que evaluaban, eran de múltiple opción, con respuesta única. También en estos casos se han utilizado las herramientas que se han creado en este proyecto, aunque no el patrón de corrección OpenMath.

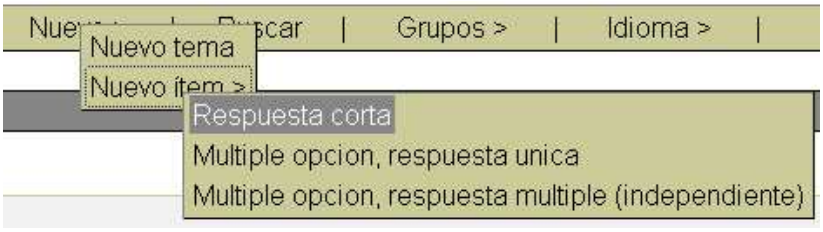


Figura 5.3: Botón de Nuevo Ítem (Respuesta Corta)

Al crear un ítem de respuesta corta en SIETTE, en particular en el editor de tests AthoS, hay que rellenar una información mínima para que el ítem sea válido. Estas son el nombre del ítem, su enunciado, al menos un patrón que sea correcto, un ejemplo válido por si el alumno erra en su respuesta y establecer si éste ítem es activo o no. Esto último sirve para que el ítem se incluya en un tests si éste se ha definido sobre el tema al que pertenece el ítem o sobre un ancestro. Normalmente queremos que nuestros ítems se incluyan en los tests que definamos, así que todos se crearon como activos.

Empezando por el título, éste puede ser el que queramos. Para este caso de uso hemos intentado poner nombres lo más descriptivos posibles.

Para el campo enunciado tenemos varias posibilidades. El profesor puede escribir directamente en el cuadro de texto que le proporciona la interfaz del editor de tests, introduciendo él directamente el texto, ya sea texto plano, o texto formateado con HTML, si conoce el lenguaje, claro está. Pero en éste caso, la capacidad expresiva es escasa, sobre todo si no se conoce mucho HTML, y la posibilidad de escribir algo parecido a una fórmula matemática es bastante remota. Para ello hemos creado el editor OpenSiette en su versión enunciados. Al haber establecido en los parámetros de la asignatura que el editor general sea éste, al pulsar el botón Editor que se encuentra a la izquierda de la caja de texto, se lanzará la interfaz de editores externos que engloba al editor OpenSiette. Ésta interfaz cuenta con, además de la inclusión del editor, varios botones para Guardar los cambios, Salir sin guardar, o Cambiar de editor.

Partiendo de este texto en blanco, a continuación vamos a ejemplificar el uso del editor OpenSiette para enunciados. Más adelante explicaremos que también funciona, claro está, para cuando ya se ha introducido texto y se quiere volver a editar.

Inicialmente el editor muestra una interfaz simple, dividida en dos partes bien diferenciadas. Por un lado el área de edición, y encima de ésta, la barra de herramientas y barra de menús. La barra de menús es bastante simple, y sólo engloba dos submenús, el de Archivo y el de Opciones. El submenú de Archivo tiene las opciones de Salir y de Guardar Cambios, que aunque están replicadas por la interfaz de editores externos de AthoS, son útiles para la interfaz de TEDI, que no dispone de ellos. El submenú de opciones es muy importante, pues habilita o deshabilita la aparición al final del enunciado, el código HTML necesario para mostrar al alumno el Editor de Entrada para poder responder. De éste tema hablaremos más adelante cuando se comenten la opción de ítem auto-respondidos.

Figura 5.4: OpenSiette. Menús

La barra de herramientas muestra los botones básicos para la edición de un texto y otros más específicos. Ésta incluye Deshacer y Rehacer, Alineado Centrado, Izquierda, Derecha y Justificado, Color de fuente Rojo, Azul y Negro, Negrita, Cursiva y Subrayado. Salvo los botones de Deshacer y Rehacer, que sirven para deshacer y rehacer cambios, el resto de acciones repercuten en el texto seleccionado cuando se pincha sobre ellas. Si no hay ningún texto seleccionado, donde comienza el cursor se empezará a aplicar el formato seleccionado. A un texto se le puede aplicar cualquiera de estas características, teniendo en cuenta que tanto la Alineación como el Color tienen valores que son excluyentes. Se pueden generar enunciados vistosos con un uso inteligente de estas opciones.

Figura 5.5: OpenSiette. Barra de herramientas

El otro botón que no se ha comentado y que es de los más importantes es el que inserta fórmulas matemáticas en el texto. Este editor se creó con el propósito de ser una herramienta WYSIWYG (What You See Is What You Get, Lo que ves es lo que obtienes), con lo cual, cualquier texto editado o fórmula matemática insertada ha de verse igual en el editor que luego en su correspondiente texto en HTML. Al utilizar este botón se crea un recuadro para la fórmula en el área de edición que contiene el texto "formula" resaltado. Pinchando en este texto se lanza el editor de fórmulas, un editor específico que permite editar cualquier fórmula incluyendo un amplio abanico de operadores y funciones matemáticas.

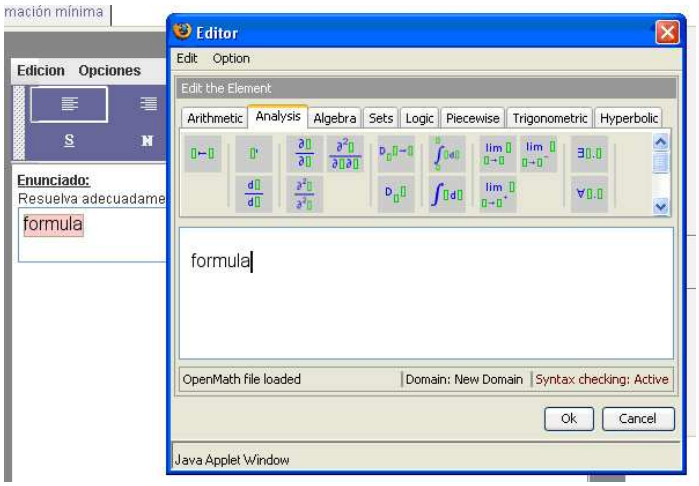


Figura 5.6: OpenSiette. Editando una fórmula

Este editor es muy parecido al que los alumnos verán al responder, o el que verá el profesor cuando edite un patrón OpenMath, pero con la diferencia de que éstos tienen un dominio restringido de operadores y funciones, mientras que éste incluye desde el campo de la Aritmética y la Lógica, al Análisis Funcional o las Funciones Hiperbólicas, pasando por las Funciones a trozos o las Matrices. Todas estas posibilidades aumentan la riqueza del enunciado.

Una vez editada la fórmula, se pulsa OK y el resultado se incluye en éste área que se definía antes. Siempre se puede volver a editar simplemente haciendo clic con el ratón sobre la fórmula. Después de la fórmula se puede seguir insertando texto, u otras fórmulas si se desea. Para este ejemplo, hemos terminado con la frase “Para responder utiliza el siguiente applet”. Esto lo hemos hecho en todos los ítems de respuesta corta que utilizarán el patrón OpenMath. Ahora simplemente, en el menú opciones bastará con seleccionar la casilla de Applet Respuesta y automáticamente, el editor generará el código HTML necesario para representar este enunciado junto al código HTML necesario para mostrar el applet de respuesta del alumno.

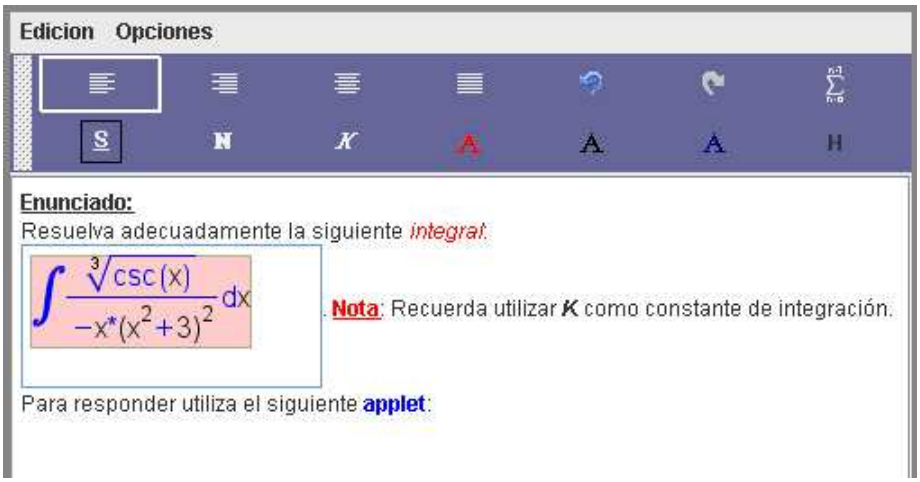


Figura 5.7: OpenSiette. Texto completamente editado

El resultado visto en HTML es, tal como se esperaba, idéntico al mostrado por el editor, junto al editor de respuesta para el alumno. Una vez establecido este texto, se puede volver a editar por el mismo procedimiento. Este editor es bidireccional. Devuelve código HTML que representa lo que se está editando y acepta también código HTML que convierte en la visualización que se edita. Como se comentó en el capítulo 4, acepta HTML de cualquier versión aunque devuelve HTML 4.0, haciendo las transformaciones necesarias. El conjunto de tags de HTML que acepta se encuentra en los anexos.

Figura 5.8: Vista del enunciado en HTML

Este código en HTML puede editarse también en el área de texto que proporciona la interfaz del editor de tests, pero no es aconsejable ya que las fórmulas matemáticas están codificadas para que sean representables en este lenguaje.

Pues una vez introducido el enunciado, pasemos a la definición del patrón correcto. Podemos añadir cuantos patrones queramos, siempre y cuando uno de ellos sea correcto. Para añadir patrones pulsamos en el botón adecuado del área de patrones de la interfaz del editor de tests y se nos mostrará un pequeño cuadro donde podemos establecer el texto del patrón así como su refuerzo.

Para el correcto funcionamiento del editor OpenSiette en su versión de respuestas (del patrón), debemos escribir el texto "<OMOBJ" como texto del patrón. Este sencillo paso indicará a OpenSiette qué interfaz desplegar. Cuando se ha creado el patrón, se pulsa el botón editar y otra vez la interfaz de editores externos de AthoS lanzará el editor OpenSiette, en su versión de respuestas. En este caso presenta la misma vista que el editor de respuestas para el alumno. Está claro que el profesor deberá utilizar la misma herramienta para definir una respuesta correcta que la que utilizará el alumno para definir la suya. Ambos comparten el mismo dominio de funciones y operadores matemáticos.

Figura 5.9: OpenSiette Respuestas

Una vez esté cargado el editor, el profesor simplemente ha de escribir una de las posibles respuestas al enunciado que ha planteado. No existe en este caso la necesidad de crear varios patrones equivalentes para tratar de captar el rango de respuestas

equivalentes que se presupone que los alumnos responderán, ya que cómo se ha comentado, el patrón de corrección OpenMath implementado en este proyecto se encarga de que esto no haga falta, utilizando la igualdad semántica para comparar respuestas de los alumnos y respuestas establecidas por los profesores.

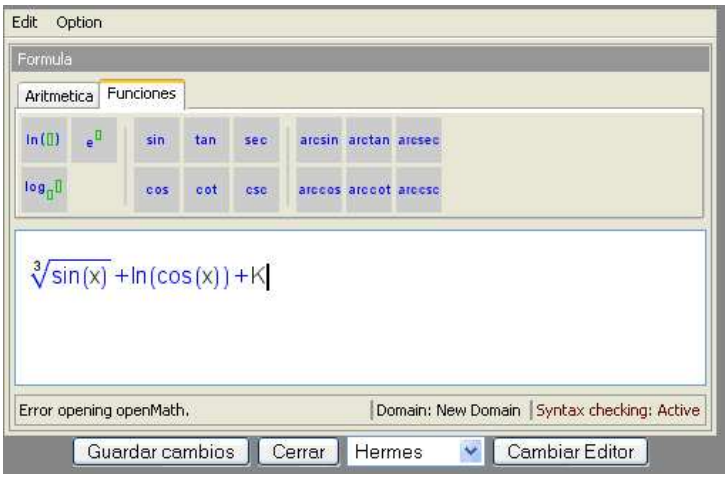


Figura 5.10: OpenSiette. Editando el patrón.

Una vez creada la respuesta correcta, el patrón se genera en código OpenMath. Éste código se puede editar de nuevo, ya que al igual que en su versión para enunciados, este editor es bidireccional.

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0"
cdbase="http://www.openmath.org/cd">
   <OMA>
      <OMS cd="arith1" name="plus"/>
      <OMA>
         <OMS cd="arith1" name="plus"/>
         <OMA>
            <OMS cd="arith1" name="root"/>
            <OMA>
               <OMS cd="transc1" name="sin"/>
               <OMV name="x"/>
            </OMA>
            <OMI>3</OMI>
         </OMA>
         <OMA>
            <OMS cd="transc1" name="ln"/>
            <OMA>
               <OMS cd="transc1" name="cos"/>
               <OMV name="x"/>
            </OMA>
         </OMA>
      </OMA>
      <OMV name="K"/>
   </OMA>
</OMOBJ>
```

Figura 5.11: Código OpenMath generado

Por último, todo ítem de respuesta corta necesita de un ejemplo válido. Aquí podemos optar por editar con OpenSiette en su versión enunciados un pequeño texto en HTML que sólo incluya la respuesta correcta.

Al terminar este proceso, si todos los campos han sido rellenados correctamente y el ítem es válido, se inserta en el tema correspondiente. Todavía quedan un par de detalles por especificar. Primero, hay que cambiar el tipo de patrón y escoger la opción Patrón OpenMath de la lista desplegable de valores posibles. Y segundo, hay que establecer que el ítem es auto-respondido, es decir, que el applet de inserción de respuestas que se le ofrece al alumno es el que va a comunicar a SIETTE la respuesta del mismo, y que no tiene que buscarla a los campos de formulario que normalmente muestra. Para esto se cambia la propiedad Auto corrección al valor Java Applet.

Una vez realizados estos pasos, tenemos un ítem de contenido matemático listo para ser integrado en cualquier tests de los que pueden crearse con SIETTE.

Hay que mencionar que las ampliaciones y herramientas desarrolladas en este proyecto no sólo permiten crear ítems de respuesta corta con contenido matemático. También es posible crear ítems de múltiple opción con respuesta única, en las que tanto sus enunciados como sus respuestas sean código HTML que incluya fórmulas matemáticas para mejorar la expresividad y la comprensión del mismo. Estos ítems ya no son auto-respondidos, si no que el alumno elige la respuesta de entre los campos del formulario y SIETTE se encarga de registrar qué respuesta escogió.

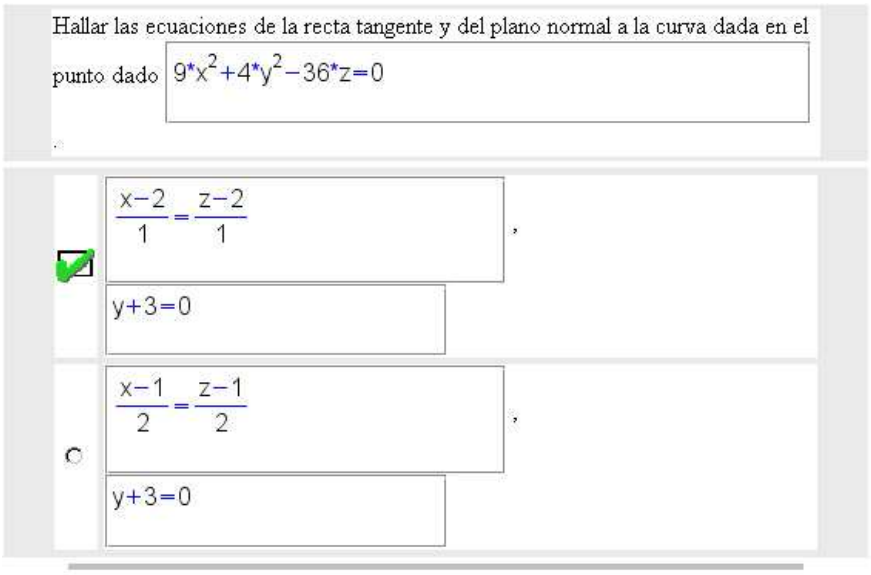


Figura 5.12: Pregunta de Múltiple Opción

5.2 La visión del alumno

En la sección anterior hemos visto como las herramientas implementadas en este proyecto ayudaban al profesor a crear asignaturas (y sus consiguientes tests) con contenido matemático evaluables en el entorno SIETTE. En ésta veremos como un alumno utiliza estas herramientas para responder a estas preguntas de respuesta abierta y cómo SIETTE compara estas respuestas con las establecidas previamente por el profesor.

Inicialmente cualquier persona con acceso a Internet puede ser alumno en SIETTE. Basta con rellenar los campos para crear una nueva cuenta de usuario con la que poder acceder al sistema. Una vez creada, accediendo, se pueden seleccionar cualquiera de las asignaturas que están activas en ese momento. Dentro de ellas, se seleccionará alguno de los tests que haya dentro de la misma, para poder comenzar la evaluación.

Dentro de nuestra asignatura hemos creado dos tests, uno de todo el contenido y otro específico de integrales. Los siguientes ejemplos los hemos tomado del que es sólo de integrales. El listado completo de ítems se puede encontrar en los anexos.

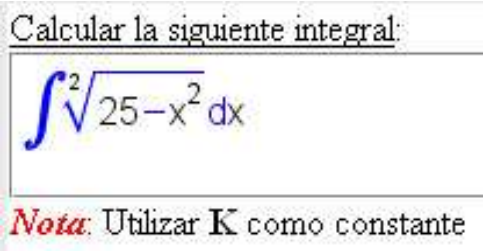


Figura 5.13: Enunciado

El alumno se encuentra en cada pregunta el enunciado, acompañado del applet de edición de respuestas. Éste presenta la misma interfaz que el applet de edición de respuestas del profesor, pero son herramientas diferentes. El alumno dispone de un área de edición, donde puede escribir, y a la vez, usar los operadores y funciones que se le ofrecen para poder ir construyendo la fórmula.

Mientras se va editando se activa la opción de corrección de sintaxis, que avisa con un subrayado rojo en sierra los componentes de la fórmula que no son correctos ya sea porque faltan partes de la misma o porque no se han usado los operadores en su ámbito adecuado. Una vez la fórmula es correcta se pulsa el botón de "Enviar Respuesta". Igualmente se puede pulsar si la respuesta tenía errores de sintaxis o si el alumno la dejo en blanco, pero será una respuesta incorrecta.

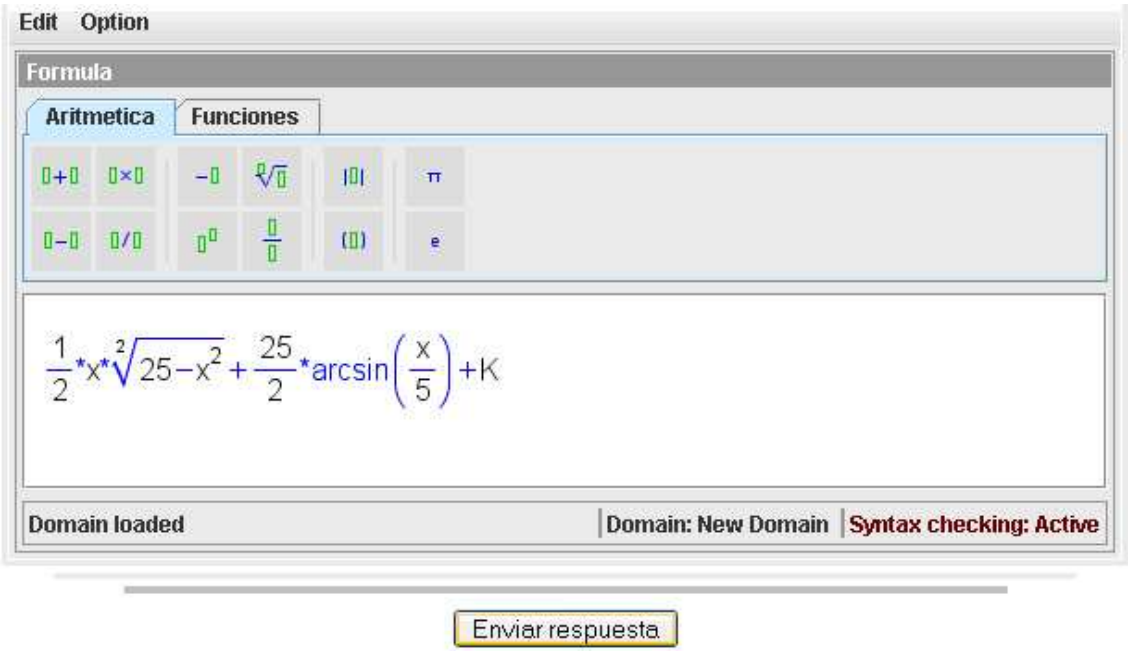
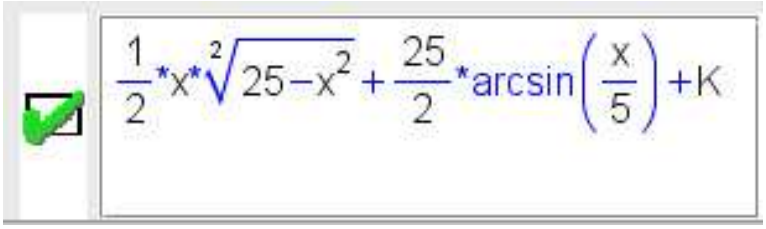


Figura 5.14: Applet de respuesta del alumno

Una vez se ha pulsado el botón, SIETTE recoge la respuesta del alumno del applet de respuesta y lo manda por los canales de comunicación que tiene establecidos al módulo de corrección. En éste momento es cuando se activa el patrón de corrección OpenMath. Éste patrón, extrae de la respuesta del alumno (que recordando lo que se vio en el capítulo anterior, está compuesta de la representación visual en HTML de la respuesta además de la propia respuesta cifrada) el código OpenMath y lo compara con el establecido por el profesor. Utilizando la igualdad semántica a través del evaluador numérico, se determina la corrección de la respuesta. Luego, la respuesta se muestra, mostrando si es correcta o no. Ésta respuesta es en realidad la representación visual de lo que en realidad respondió el alumno, y es una de las variaciones del applet de Wiris, que permite mostrar fórmulas sin que sean editables y las barras de herramientas o menús del editor.



5.15: Visualización respuesta correcta del alumno

Si la respuesta no fuese correcta, se mostraría el ejemplo válido que se definió. En este caso el test lo tenemos definido para que muestre la corrección justo después de cada pregunta, pero esto se puede cambiar.

Hemos dicho que el patrón de corrección OpenMath se basa en la igualdad semántica de las fórmulas. Para comprobarlo vamos a mostrar unas cuantas preguntas de este test, en las que la forma de expresar la misma fórmula por parte del profesor difiere a cómo la respondió el alumno.

Primero vamos a probarlo con un cambio sencillo, como aplicar la propiedad conmutativa de la suma. Para un enunciado como este:

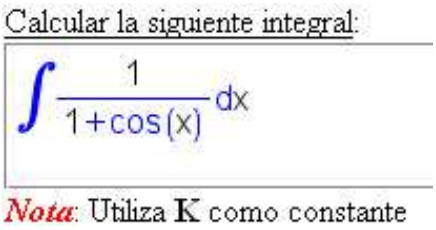


Figura 5.16: Enunciado pregunta I

El profesor definió la siguiente respuesta como correcta:

$$-\cot(x)+\csc(x)+K$$

Figura 5.17: Patrón de pregunta I

Y el alumno respondió correctamente con la siguiente:

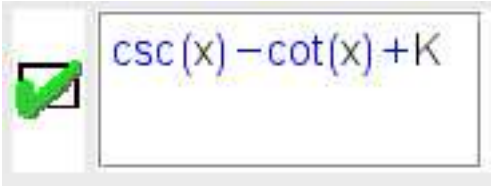


Figura 5.18: Respuesta correcta a pregunta I

Lo mismo ocurre en estos dos nuevos casos, donde los cambios son más notables y aún así, las respuestas son equivalentes y correctas ambas.

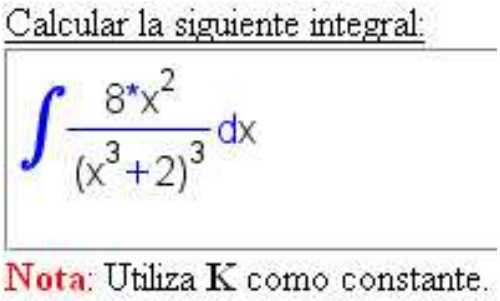


Figura 5.19: Enunciado pregunta II

$$-\frac{4}{3}*\frac{1}{(x^3+2)^2}+K$$

Figura 5.20: Patrón de pregunta II

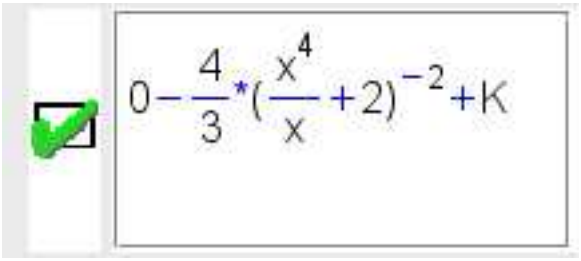


Figura 5.21: Respuesta correcta a pregunta II

El segundo caso:

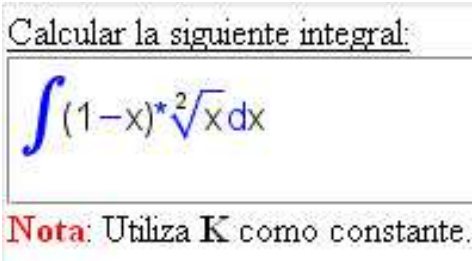


Figura 5.22: Enunciado a pregunta III

$$\frac{2}{3}*x^{\frac{3}{2}}-\frac{2}{5}*x^{\frac{5}{2}}+K$$

Figura 5.23: Patrón de pregunta III

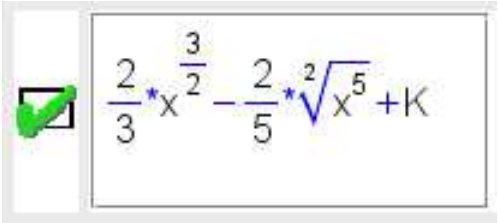


Figura 5.24: Respuesta correcta a pregunta III

Con esto se muestra la versatilidad del sistema implementado para ampliar la funcionalidad de SIETTE con la capacidad de poder crear y evaluar contenido matemático de una forma automática y visual.

Capítulo 6
Conclusiones y futuras mejoras

6 Conclusiones y futuras mejoras

Este ha sido un proyecto de integración de tecnologías. Desde un principio el enfoque fue de conjugar la potencia de Siette con el software Wiris y la utilización del nuevo estándar OpenMath para la representación de contenido matemático en un entorno informatizado.

Por esto mismo, gran parte de la labor ha sido la comprensión de las funcionalidades de estas tecnologías y sus limitaciones, y las características intrínsecas de cada una de ellas que permitirían sacarles el máximo partido para finalmente crear una solución útil y funcional al problema que se planteaba.

Donde más se aprecia este aspecto de integración de tecnologías es en el plano del proceso software, donde el conocimiento de las verdaderas posibilidades de interacción de los diferentes elementos era primordial, pues un mal diseño o un desconocimiento parcial de esta podrían llevar a la no consecución de los objetivos tal como se hubieran planteado en un principio.

Para este proyecto se ha conjugado los applets de visualización de Wiris, que utilizan OpenMath, integrándolos en Siette. Se ha mejorado Siette para que acepte la corrección de un nuevo tipo de respuestas que permiten más capacidad expresiva cuando de contenidos matemáticos se tratan. Se han creado decenas de ejemplos en el sitio público de Siette, pues desde el principio este proyecto fue pensado para su utilización pública. Éste aspecto fue uno de los condicionantes más importantes.

No obstante, aunque la solución creada completa todos los objetivos pensados, existen ciertos aspectos que están abiertos a mejoras y que pasamos a comentar:

Con respecto a la corrección de las respuestas de los alumnos, sería posible ampliar el espectro de operadores y funciones matemáticas con las que pueden responder, ampliando también el rango de respuestas que el profesor puede definir, mejorando así la diversidad de contenidos.

Otra mejora, como se vio en la solución que daba LeActiveMath al mismo problema, y dado que OpenMath está diseñado para el intercambio de estos objetos entre diferentes sistemas, sería la integración de un CAS con Siette. Esta conexión permitiría el envío de las respuestas del alumno directamente junto a la del profesor, donde se resolverían. Como se comentó existen varios PhraseBooks, o traductores de OpenMath a lenguaje interno para muchos CAS. A través de un servicio web, de XML RPC o de algún otro sistema de RPC se podría realizar esta comunicación. Desgraciadamente, la existencia de estos PhraseBooks para CAS no comerciales y de libre distribución es casi nula.

Bibliografía

Bibliografía

Bibliografía consultada

1. Tesis Doctoral: Un modelo de evaluación cognitiva basado en Tests Adaptativos para el diagnóstico en Sistemas Tutores Inteligentes. Eduardo Guzman de los Riscos. Octubre 2005.
2. Cálculo Diferencial e Integral. Frank Ayres Jr, Elliot Mendelson. McGrawHill
3. Nociones de Algebra y Trigonometría. J. Rey Pastor, P. Puig Adam. Nuevas gráficas. 1963
4. Tratado Elemental de Trigonometría. Miguel Aguayo y Millan. Ediciones Universidad de León. D.L. 2000
5. The OpenMath Standard. S. Buswel et alter. Junio 2004.
6. Interactivity of Exercises in ActiveMath. Goguadze G, Gonzalez Palomo A, Mellis E. LeActiveMath Project.
7. Using Dialogue to Learn Math in the LeActiveMath Project. Callaway C et Al. LeActiveMath Project.
8. ActiveMath: An Intelligent Tutoring System for Mathematics. Mellis, E., Sieckman G. German Research Institute for Artificial Intelligence.
9. Database System Concepts. Silberschatz, Korth, Sudarshan. McGrawHill. 2004
10. Teach Yourself J2EE in 21 Days. Dan Haywood et al. Sams. 2004.
11. JAVA 2. Manual de Usuario y tutorial. Froute Quintas, A. Ra-Ma 2005
12. XML in a Nutshell, Elliotte Rusty Harold & W. Scott Mears. O'Reilly. 2004
13. Ingeniería del software, un enfoque práctico. Roger Presuman. McGrawHill. 2006
14. Patrones de diseño : elementos de software orientado a objetos reutilizable. Erich Gamma. Addison Wesley 2005.
15. Sams teach yourself UML in 24 hours. Joseph Schmuller. Sams. 2002

Recursos web y otras referencias consultadas

16. Wiris - www.wiris.com

17. OpenMath - www.openmath.org
18. SIETTE - www.lcc.uma.es/siette
19. Applets Wiris - http://klein.activemath.org/~paul/tmp/WirisApplets
20. W3Schools - http://www.w3schools.com/
21. ALEKS - www.aleks.com
22. LeActiveMath - http://leam-calculus.activemath.org
23. Java OpenMath Library - http://pdg.cecm.sfu.ca/opnemath0.5
24. RIACA phrasebooks - http://www.riaca.win.tue.nl/products/
25. Test de Matemáticas online - http://amby.com/tests/math
26. The online Math tests Home Page - http://mathonline.missouri.edu
27. ThatQuiz - www.thatquiz.com
28. American National Standards Insititute - www.ansi.org
29. International Standards Organization - www.iso.org
30. Wikipedia España http://es.wikipedia.org
31. E-learing Europa - http://www.elearningeuropa.info
32. The DNA of e-learning. Jay Cross, Ian Hamilton - http://www.elearningworkshops.com/docs/estrategia/DNA.pdf
33. Boletin de la Sociedad de la Información - http://sociedaddelainformacion.telefonica.es
34. QTI - www.imsglobal.com
35. SCORM - www.scorm.com
36. PHP - www.php.net
37. Moodle - www.moodle.org
38. PostgreSQL - www.postgresql.org
39. Regex Java - http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html
40. Ant - http://ant.apache.org/
41. Tomcat - http://tomcat.apache.org/
42. CVS - http://www.nongnu.org/cvs/
43. Eclipse - http://www.eclipse.org/
44. SVQ - www.w3.org/Graphics/SVG
45. XUL - www.xulplanet.org
46. OMG - www.omg.org
47. CSS Colors - http://www.w3schools.com/css/css_colornames.asp
48. Intralearn - www.intralearn.com

49. WebAssessor - www.webassessor.com
50. WebCT - www.webct.com
51. Terranova CAT - www.ctb.com
52. CAT Global - www.catglobal.com
53. Quanta - www.quanta.co.uk/
54. INSPIRE - An INtelligent System for Personalized Instruction in a Remote Environment (2002). Papanikolaou, K. A., Grigoriadou, M., Kornikolakis, H. y Magoulas, G. D. (2003).
55. Skate - Chua Abdullah, 2003
56. PASS - Personalized Assessment Module (Gouli et al) 2002
57. Alice - Adaptive Link Insertion in Concept-based Educational System. Kavcic et al. 2002
58. Andes - www.andes.pitt.edu
59. Polas - http://portal.acm.org/citation.cfm?id=331700

Anexos

Anexo A

Lista de tags de HTML soportados por el Applet

b	Este elemento muestra en negrita el texto que engloba. Su anidación no anula ni aumenta el efecto. Su uso está obsoleto en favor de los estilos CSS aunque no está descatalogado.
strong	Este elemento, del estándar 4.01 resalta en negrita el texto que englobe, aunque depende de la implementación del navegador. Su anidación no anula ni aumenta el efecto.
i	Este elemento muestra en cursiva el texto al que engloba. Su anidación no anula ni aumenta el efecto. Su uso está obsoleto a favor de los estilos CSS, aunque no está descatalogado.
u	Este elemento muestra subrayado el texto al que engloba. Su anidación no anula ni aumenta el efecto. Descatalogado desde la versión 4.0 a favor del uso de estilos CSS.
span	Este elemento es estructurante. Sirve para agrupar elementos con unas mismas características, que se pueden establecer a través de estilos CSS. La anidación de varios elementos span puede aumentar o anular el efecto de estas propiedades si los elementos más internos las redefinen o las redeclaran.
font	Este elemento cambia la apariencia del texto al que engloba. Puede llevar tres atributos: size (tamaño), face(familia de la fuente) y color. El uso anidado de este elemento puede anular o aumentar los efectos del formateado si los elementos interiores redefinen o redeclaran los valores de estos atributos. Su uso está descatalogado desde la versión 4.0 a favor de los estilos CSS.
p	Este elemento representa un párrafo. Añade un espacio antes y después del texto que engloba. Permite la definición de estilos entre sus atributos. Su uso anidado puede anular o aumentar el efecto de estos estilos si los elementos interiores redefinen los estilos CSS.
br	Este elemento representa un salto de línea.
applet	Este elemento representa un applet de Java. En sus atributos deben encontrarse archive (el fichero donde está la clase), code (el nombre de la

	clase), codebase (el URL relativo al fichero), además de atributos de tamaño o alineación.
param	Este elemento representa un parámetro del applet. Su uso esta restringido a este contexto. Posee los atributos name y value.

Anexo B

Lista de propiedades CSS2 soportadas por el applet de Edición de Enunciados

text-align	Alinea el texto dentro de un elemento. Sus posibles valores son: left, right, center y justify.
text-decoration	Añade una decoración al texto. Sus posibles valores son: none (ninguna), underline (subrayado), overline (suprarayado), line-through (tachado), blink (parpadeante).
font-weight	Establece el grosor de la fuente. Sus valores pueden ser normal, bold (en negrita), bolder (más negrita que el valor que ya tenia), lighter (menos negrita que el valor que ya tenía) o un valor del 100 al 900 (en incrementos de 100).
font-style	Establece el estilo de la fuente. Sus valores pueden ser normal, oblique (oblicuo) e italic (cursiva) .
color	Establece el color del texto. Su valor puede ser cualquier valor RGB hexadecimal o un nombre preestablecido (lista)

Anexo C

Manual de Usuario editor OpenSiette

El editor OpenSiette se compone de un área de edición, una barra de herramientas y un menú.

El menú se compone de dos submenús, Archivo y Opciones.

Submenú Archivo: Se compone de dos opciones: Guardar y Salir. Pulsando la opción Guardar se copian los cambios hechos al campo del Editor de Tests y se cierra el editor. Pulsando Salir simplemente se cierra el editor, perdiéndose los cambios realizados.

Submenú Opciones: Se compone sólo de una opción, Applet Respuesta. La activación de esta opción añade o elimina del final del enunciado el código HTML necesario para mostrar el applet de respuestas del alumno, el que le va a permitir responder a lo que el profesor está estableciendo en el enunciado. Este applet tiene sentido en las preguntas de respuesta libre, pero no en la de respuesta múltiple.

La barra de herramientas se compone de una serie de botones:

Deshacer: Deshace la última acción, si es posible.

Rehacer: Rehace la última acción deshecha, si es posible.

Alineación Justificada: Formatea el texto seleccionado con una alineación justificada. Se aplica a todo un párrafo, esté completamente seleccionado o no.

Alineación Centrada: Formatea el texto seleccionado con una alienación centrada.

Alineación Derecha: Formatea el texto seleccionado con una alineación a la derecha.

Alineación Izquierda: Formatea el texto seleccionado con una alineación a la derecha.

Negrita: Formatea el texto seleccionado poniéndolo en negrita.

Cursiva: Formatea el texto seleccionado poniéndolo en cursiva.

Subrayado: Formatea el texto seleccionado subrayándolo.

Color Azul: Formatea el texto seleccionado poniéndolo en color azul.

Color Rojo: Formatea el texto seleccionado poniéndolo en color rojo,

Fórmula Matemática: Inserta una fórmula matemática en el texto.

Todas las acciones de formateo del texto se pueden aplicar sobre puntos del área de edición. El resultado de esto es que cualquier texto insertado en ese punto tomará el estilo seleccionado.

El límite de la pila de acciones por hacer o deshacer es de 20.

La inserción de una fórmula dará como resultado la aparición de un cuadro donde se incluye un área rosada, que es donde realmente está la fórmula. Haciendo clic sobre ésta, se activa el applet de edición. La fórmula se va construyendo utilizando los operadores y fórmulas matemáticas que se ofrece. A través de teclado se pueden incluir signos como +, -, / ó *, pero no otros como =, que tienen su propio símbolo.