INGENIERÍA INFORMÁTICA

Curso Académico 2005 / 2006

Proyecto de Fin de Carrera

MODELADO Y GESTIÓN DE LA VARIABILIDAD EN SISTEMAS SOFTWARE

Autor: Alejandro Sánchez Valdezate
Tutor: Rafael Capilla

INDICE

1. Introducción

La tendencia actual de las empresas consiste en recortar costes de producción de software y a la vez incrementar la calidad del mismo en el menor tiempo posible. Por otra parte, dado que dentro del ámbito de negocio de las empresas de software es muy frecuente desarrollar varias veces los mismos componentes para distintos productos, el uso de componentes reutilizables es uno de los factores clave en el ahorro de costes y de tiempo. Asimismo, el desarrollo ágil de componentes software para productos similares con características comunes es uno de los aspectos clave a considerar. Así por ejemplo, los sistemas Web es un ejemplo claro de construcción de productos con similares características en periodos cortos de tiempo y motivados por requisitos de mercado.

Por estos motivos, el objetivo principal de este Proyecto Fin de Carrera consiste en la creación y mantenimiento de familias de productos de características semejantes que permitan elaborar todas las combinaciones de puntos de variación con el fin de obtener todos los productos posibles a partir de una línea de productos concreta.

Además, estudiaríamos los aspectos de variabilidad en los sistemas software para elaborar distintas familias de productos con similares características, a través de la elección de piezas (puntos de variación) que compondrán la familia de productos.

El análisis de variabilidad realizado nos permite conocer el número total de productos obtenibles a partir de una línea de productos concreta, así como parametrizar la línea de productos de cara a definir restricciones, tal y como ocurre en el mundo empresarial, por lo que este proyecto es directamente aplicable a los árboles de decisión, cadenas de montaje y estrategias de mercado, de forma que se obtenga un beneficio directo de la construcción y uso de componentes reutilizables.

2. Estado del Arte

Antes de describir los aspectos de variabilidad de los sistemas en relación con las arquitecturas de líneas de productos, vamos a mencionar las bases teóricas que fundamentan las arquitecturas software, las cuales resultan imprescindibles en la construcción de sistemas similares.

2.1. Patrones de Diseño Software

La similitud de muchas de las soluciones empleadas en los sistemas software actuales y los problemas de mantenimiento existentes hacen que dichas soluciones se repitan en los distintos productos software, tanto a nivel de diseño como de implementación. De esta manera, los productos creados comparten elementos y soluciones comunes. Por este motivo diversos autores han propuesto el uso de patrones de diseño y patrones arquitectónicos como solución a problemas que se repiten de forma habitual.

El uso de un patrón de diseño para dirigir la creación de software hace posible que una empresa realice de manera eficiente una actividad determinada (aquella en la que está especializada), poniendo especial interés en los elementos definidos en el patrón a seguir. En términos generales, un patrón de diseño software constituye una solución probada para un problema recurrente.

Según Buschmann [Buschmann et. al., 1996], un patrón desde el punto de vista de la arquitectura software describe un problema de diseño particular que se repite en un contexto de diseño específico, y presenta un esquema genérico debidamente probado para su solución. El esquema de la solución está especificado por la descripción de sus componentes, sus responsabilidades y relaciones, y la forma en la que colaboran entre sí. Los estilos arquitectónicos consisten en la descripción de los tipos de componentes y un patrón de su control de ejecución y/o transferencia de datos.

Tabla 1. Clasificación de patrones de diseño por su propósito

Tipo	*Descripción*	*Ejemplos*
De creación	Se refieren a la creación de instancias de las clases	Abstract Factory, Builder, Factory,Prototype, Singleton
Estructurales	Se refieren a las relaciones entre clases u objetos	Adapter, Bridge, Composite Method, Decorator, Facade, Flyweight, Proxy.
De comportamiento	Se refieren a la interacción y cooperación entre clases	Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor,

Tabla 2. Clasificación de patrones de diseño por su ámbito

Tipo	*Descripción*	*Ejemplos*
De clases	Se refieren a las relaciones de herencia (estáticas entre clases y subclases).	Adapter (clases), Interpreter, Template Method, Factory Method.
De objetos	Se refieren a relaciones de uso (dinámicas) entre objetos	Abstract Factory, Builder, Prototype, Singleton, Adapter (object), Bridge, Composite, Decorator, Façade, Flyweight, Proxy, Chain of Responsibility, Command, Iterator, Medaitor, Momento, Observer, State, Strategy, Visitor,

2.2. Concepto de Arquitectura Software

Las arquitecturas de software son una disciplina de reciente creación dentro de la ingeniería del software. Una arquitectura software constituye una solución de diseño para el sistema o sistemas software que se pretenden construir. La construcción de arquitecturas de software es una tarea compleja que se desarrolla de forma manual dado que es un proceso creativo asociado al diseñador o arquitecto (i.e.: un ingeniero software). Algunas definiciones del concepto de arquitectura del software son las siguientes:

“Una arquitectura software implica la descripción de los elementos a partir de los cuales se construye un sistema, las interacciones entre ellos, un conjunto de patrones que guían su composición, y restricciones entre estos patrones” [Shaw y Garlan, 1996].

“La arquitectura software de sistema software es la estructura o estructuras del sistema, que comprende componentes software, las propiedades externamente visibles de estos componentes y las relaciones entre ellos" [Bass et al., 1998b]”

COMPONENTES

Un componente es una unidad elemental de composición para la construcción de aplicaciones software mediante su adaptación y ensamblado. Estas unidades se pueden clasificar según los diferentes contextos en los que la palabra componente se utiliza, ya que, como indica Hopkins: “Mientras que el concepto de componente software es conocido virtualmente desde los inicios de la ingeniería del software, la problemática y aspectos prácticos relativos a los mismos han evolucionado continuamente a lo largo del tiempo” [Hopkins, 2000].

A continuación se recogen diferentes definiciones del término componente, aunque siempre desde la perspectiva del desarrollo basado en componentes y en relación con la reutilización del software.

- Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio [Szyperski y Pfister, 1997].

- Un componente es una unidad de composición con sus interfaces contractuales especificadas y únicamente con dependencias explícitas del contexto. Un componente software puede ser distribuido de forma independiente y puede ser objeto de composición por terceras partes [Weck et al., 1997].

- Un componente es un paquete software que ofrece servicios a través de sus interfaces

- Un conjunto de componentes atómicos simultáneamente distribuidos. Un componente atómico es un 'módulo' más un conjunto de 'recursos'. Un módulo es un conjunto de clases y posiblemente otras construcciones no orientadas al objeto, tales como procedimientos o funciones. Un recurso es una colección estática de elementos tipados (que parametrizan al componente) [Szyperski, 1998].

- Un componente debe cumplir [Meyer, 1999]:

    - Puede ser usado por otros elementos software.

    - Puede ser usado por clientes sin intervención del desarrollador del componente.

- Incluye una especificación de todas sus dependencias.
- Incluye una especificación de la funcionalidad que él ofrece. Es usable sólo en la base de sus especificaciones.
- Se puede componer con otros componentes.
- Se puede integrar en un sistema de forma rápida y sin conflictos.

CONECTORES

Los conectores es un tipo de elemento arquitectónico que modela interacciones entre componentes y las reglas que gobiernan estas interacciones. Nos Existen diferentes tipos de conectores en función del tipo de interacción. Estos son:

- Interacciones simples:
  - llamada a procedimiento
  - acceso a variables compartidas
- Interacciones complejas y semánticamente ricas:
  - protocolos cliente-servidor
  - protocolos de acceso a bases de datos
  - eventos multicast asíncronos
  - flujos de datos en pipe-line

La base de las arquitecturas software la constituyen principalmente, los patrones de diseño, los estilos arquitectónicos (patrones arquitectónicos) y los modelos de referencia.

Estilos arquitectónicos: Consisten en la descripción de tipos de componentes y de los patrones de interacción entre ellos. Se puede pensar como un conjunto de restricciones sobre una arquitectura que a su vez define una familia de arquitecturas que las satisfacen [Bass et al., 1998b]. Según [Shaw y Garlan, 1996] caracteriza una familia de sistemas que están relacionados por compartir propiedades estructurales y semánticas.

2.2.1. Un ejemplo: Estilo de Capas o Niveles

Este estilo se basa en la descomposición de una aplicación en un conjunto de capas independientes y ordenadas jerárquicamente. Cada nivel o capa usa los servicios de la capa inmediatamente inferior y ofrece servicios a la capa inmediatamente superior. Un ejemplo para muchos conocidos es la torre de niveles OSI (Open System Interconnections). Fue diseñado con el fin de conseguir la definición de un conjunto de normas que permitieran interconectar diferentes equipos, posibilitando de esta forma la comunicación entre ellos.

Tabla 3. Ventajas e inconvenientes del estilo de capas

VENTAJAS	INCONVENIENTES
Se puede comprender el comportamiento global del sistema como una simple composición del comportamiento de los filtros.	Frecuentemente derivan en organizaciones batch.
Soporta la reutilización de los filtros	No adecuados para procesamiento interactivo

Sistemas fáciles de mantener y mejorar.	Fuerzan denominador común en transmisión (parse y unparse los datos): fuente de ineficiencia y complejidad.
Permiten análisis (rendimiento, interbloqueos)	Si el número de niveles es excesivo, puede ser muy ineficiente.
Soportan la ejecución concurrente	Trabajo innecesario de paso de argumentos entre niveles.
Reutilización de un mismo nivel en varias aplicaciones.	Si hay pocos niveles tenemos el diseño poco organizado. Si hay excesivos niveles el sistema es muy complejo e ineficiente.
Permite la estandarización	
El cambio de un nivel no afecta al resto. Pero un mal diseño o un cambio importante de funcionalidad pueden forzar cambios que se transmitan en cascada de un nivel a otro.	

Modelos de referencia: Constituyen una división de la funcionalidad junto con los flujos de datos entre las piezas . Por ejemplo las partes estándares de un compilador.

Arquitecturas de referencia: Constituyen el primer paso antes de obtener la arquitectura software. Una arquitectura de referencia puede entenderse como la asociación de un modelo de referencia sobre componentes software y los flujos de datos entre dichos componentes. Los componentes software implementan la funcionalidad del sistema definida en el modelo de referencia. Cada una de las divisiones de un modelo de referencia puede implementarse a través de uno o varios componentes en la arquitectura de referencia. Esta arquitectura viene a representar la huella dactilar para construir arquitecturas de software similares.

Finalmente, **la arquitectura software** se deriva a partir de la arquitectura de regencia en el proceso de construcción arquitectónico, tal y como observamos

en la figura 1. Esta arquitectura software define el sistema en términos de componentes y conectores en su definición más simple. Los conectores es un tipo de elemento arquitectónico que modela interacciones entre componentes y las reglas que gobiernan estas interacciones. Los componentes los podemos definir como una "unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio" [Szyperski y Pfister, 1997]. Es interesante ver la definición de Meyer [Meyer, 1999] sobre componentes.

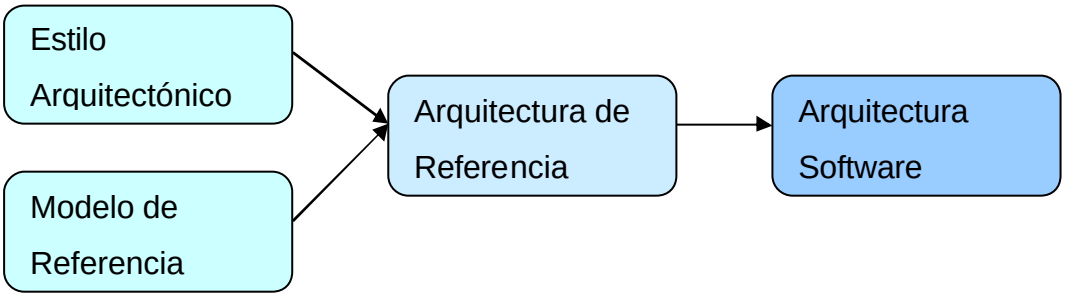


Ilustración 1. Proceso de Construcción de Arquitecturas Software

La construcción de arquitecturas software y componentes reutilizables se puede realizar principalmente desde dos enfoques. Uno de tipo "bottom-up" de abajo hacia arriba utilizando técnicas de ingeniería inversas y otro de tipo "top-down" de arriba hacia abajo utilizando procesos de análisis del dominio e ingeniería del dominio.

2.3. Procesos de Ingeniería de Dominio

Es una actividad enfocada a la construcción de componentes reutilizables que se puedan utilizar posteriormente en el desarrollo de sistemas software. Estas técnicas son utilizadas en algunos modelos de Líneas de Productos tal y como veremos más adelante.

La principal ventaja del uso de componentes reutilizables es que puedan ser empleados en la construcción de varios sistemas de software en lugar de un único sistema. En este sentido, los procesos de ingeniería del dominio producen los componentes reutilizables, mientras que el proceso denominado ingeniería de aplicaciones se encarga de construir los sistemas software a partir de dichos componentes.

Los procesos de ingeniería del dominio son procesos complejos que tratan de describir e implementar los conceptos de un dominio determinado (i.e.: por dominio entenderemos un conjunto de aplicaciones o sistemas que comparten un vocabulario común y unas características determinadas) y se componen de dos partes principales: Por un lado, un proceso denominado análisis del dominio encargado de analizar el dominio en cuestión y por otra parte un proceso de implementación consistente en el desarrollo de una arquitectura software y de los componentes reutilizables. A continuación describimos a grandes rasgos las características principales de los procesos de análisis del dominio.

2.3.1. Análisis del dominio

El concepto de análisis del dominio tiene sus orígenes en los trabajos de James M. Neighbors sobre reutilización de software basados en el paradigma Draco [Neighbors, 1984] a principios de la década de los ochenta.

Un proceso de análisis del dominio trata de identificar los objetos y operaciones de un dominio particular y obtener lo que se denomina modelo del dominio,

consistente en una descripción más o menos formal de la información obtenida durante el proceso de análisis. Según Arango [Arango et. al., 1994]. el Modelo del Dominio es un conjunto de definiciones de entidades, operaciones, eventos y relaciones clasificados que constituyen una taxonomía.

Asimismo, debido a la diversidad de métodos de análisis del dominio existentes, podemos decir que no existe acuerdo en como llevar a cabo cada una de las tareas del proceso, pero en cambio se mencionan una serie de pasos comúnmente aceptados que describen las fases fundamentales de esta técnica y que podemos resumir de la siguiente manera:

- **Determinación de los límites del dominio**: Cada método tiene una fase de preparación. Su propósito, como en cualquier proyecto, es seleccionar el dominio (involucrando estudio de factibilidad y análisis de riesgo), definir los límites y contenido del dominio, identificar datos relevantes, crear un inventario de las fuentes de información identificadas, y planificar el proyecto.

- **Recolección de Datos:** Aplicación de técnicas de adquisición de información, no necesariamente específicas del análisis de dominios.

- **Análisis de Datos:** Todos los métodos coinciden en identificar entidades, eventos, operaciones y relaciones, modularizar la información mediante descomposición de funciones y datos, o mediante análisis orientado a objetos, e identificar decisiones de diseño. También tiene lugar el análisis de los costes y beneficios.

- **Clasificación:** Los métodos difieren en los atributos usados para la clasificación. En orientación a objetos, este proceso es parte de la definición de la jerarquía de clases.

- **Generación de productos del dominio**: Una vez realizado todo el proceso el último paso constituye la generación del producto final

utilizando para ello todo el trabajo realizado durante el análisis de dominio. Algunos métodos incluyen referencias a pasos de evaluación, pero sin una especificación explícita ni objetivos de funcionamiento que faciliten la definición de un criterio de validación.

2.3.2. Ingeniería del Dominio y Variabilidad

Además de las arquitecturas software y los componentes reutilizables desarrollados con técnicas de ingeniería del dominio, existe un concepto bastante importante aplicable a las arquitecturas software: Es la personalización de componentes y la derivación de productos software similares, que es el de variabilidad del software.

La variabilidad el software trata de identificar las partes comunes y variables de los sistemas para poder construir familias de sistemas relacionados y con similares características. En relación con este concepto existen algunos métodos de clásicos e ingeniería del dominio que ponen énfasis en los aspectos de variabilidad, tal y como describimos a continuación.

FODA (Feature-Oriented Domain Analysis): En este sentido, el método FODA [Kang, 1990] La metodología Featured Oriented Domain Análisis (FODA) propone el uso de características (“features”) para describir la variabilidad de los sistemas. El concepto de variabilidad es un aspecto clave para la construcción de un conjunto de sistemas similares. La variabilidad debe reflejarse tanto en la arquitectura software como en los componentes reutilizables los cuales van a ser utilizados en la construcción de los diferentes sistemas software.

La metodología de FODA fue fundada en un sistema de modelización de conceptos para desarrollar los productos del dominio que son genéricos y extensamente aplicables dentro de un dominio. Los conceptos que modelan las

bases son la abstracción y el refinamiento. La abstracción se utiliza para crear productos genéricos del dominio. El método de FODA defiende que los usos en el dominio se deben abstraer al nivel donde no hay diferencias entre aplicaciones.

La abstracción de las aplicaciones en el dominio del uso se logra utilizando los modelos de: agregación/composición, generalización/especialización, y parametrización. El método de FODA aplica los modelos primitivos de la agregación y de la generalización para capturar las concordancias de los usos en el dominio en términos de abstracciones. Las diferencias entre las aplicaciones se traducen en refinamientos. Una abstracción se puede refinar generalmente (es decir, descompuesto o especializado) de muchas diversas maneras dependiendo del contexto en el cual se hacen los refinamientos. Los parámetros se definen para especificar únicamente el contexto para cada refinamiento específico. El resultado de este acercamiento es un producto del dominio que consiste en una colección de abstracciones y de una serie de refinamientos de cada abstracción con la parametrización. Entender qué distingue usos en un dominio es la más crítico puesto que es la base para las abstracciones, los refinamientos, y el parametrización.

El concepto orientado hacia las características de FODA se basa en el interés puesto por el método en identificar características visibles para el usuario de manera prominente o distintivas dentro de una clase de los sistemas de software relacionados. Estas características conducen a la conceptualización del sistema de los productos que definen el dominio.

2.3.3. FORM

FORM: Un método de reutilización orientado a características con referencia específica al dominio

FORM (Feature Oriented Reuse Method) es un método sistemático que busca y captura características comunes y diferencias de aplicaciones en un dominio y utiliza los resultados del análisis para desarrollar arquitecturas del dominio y componentes. Al modelo que captura los características comunes y las diferencias se le llama "modelo característica" y se utiliza para apoyar la ingeniería de los artefactos reutilizables del dominio y el desarrollo de aplicaciones usando los artefactos del dominio. Una vez que un dominio se describe y se explica en términos de "unidades comunes y diversas" , se utiliza para construir diversas configuraciones factibles de arquitecturas reutilizables. El uso de "características" es motivado por el hecho de que los clientes y los ingenieros hablan a menudo de características del producto en términos de "características que el producto tiene y/o que aporta". Se transmiten los requisitos o las funciones en términos de características y, para ellos, las características son abstracciones funcionales distintivas identificables que deben ser puestas en ejecución, probarse, entregarse y mantenerse. Este hecho, sin embargo, no ha sido apoyado ni ha sido explotado completamente por la mayoría de métodos de la tecnología de dotación lógica hasta ahora. Creemos que las características son las abstracciones que los clientes y los reveladores entienden y deben para ser los primeros objetos de la clase en el desarrollo del software.

El concepto de la “orientación a las características” no es totalmente nuevo en la tecnología de dotación lógica, y ha habido esfuerzos de basar el desarrollo del software en la noción de características, especialmente en el dominio de la telefonía. Por ejemplo, [Kang et al. 1990; Palmer y Liang 1992] se centraron en la ingeniería de requisitos “orientada a las características”; [Sloane y Holdsworth 1996] desarrolló un esquema de prueba “orientada a las características”; y [Fekete 1993; Zave 1993] introdujo un método para la especificación de las características y el análisis de la interacción. La mayoría de estos esfuerzos, sin embargo, han sido bien “ad hoc” o bien limitados a cierta fase del ciclo de vida. Sin embargo, aún debe aparecer un método comprensible y sistemático del desarrollo del software que toma la “orientación a las características” como su paradigma a través del ciclo de vida completo.

Una característica o “feature” es una abstracción funcional distintiva e identificable que puede ser empaquetada, implementada, probada, distribuida y mantenida [Kang, 1998].

La aplicación del concepto de variabilidad a las Líneas de Producto permite descubrir las partes comunes y variables de los sistemas y representarlos en la arquitectura a través de los denominados puntos de variación. En las fases del ciclo de vida previo a la obtención de los puntos de variación, consideramos a estos como explícitos y pueden estar vinculados a una variante particular.

El mantenimiento de la variabilidad debe aplicarse al nivel de diseño de componentes dentro de la ingeniería de dominio, aunque en algunos momentos debe realizarse al nivel de código. Llegar al punto intermedio y combinar ambos aspectos es lo ideal. [Slicki y Farcet, 2002].

El desarrollo de software basado en el modelo de Líneas de Producto se organiza en dos fases, ingeniería de dominio y aplicación. La ingeniería de dominio se interesa en la identificación de las partes comunes y variantes de los productos dentro de la línea de productos y la implementación de artefactos comunes. Durante la ingeniería de aplicación los productos individuales son desarrollados por selección y configuración de artefactos comunes y, donde resulte necesario, la adición de las extensiones específicas del producto.

2.4. Arquitecturas de Líneas de productos

Informalmente una línea de productos se puede ver como un conjunto de productos que están estrechamente relacionados (por su funcionalidad), que son vendidos a los mismos grupos de compradores, que son comercializados a través del mismo tipo de distribución, o que caen en el mismo rango de precios.

Las definiciones formales de líneas de producto (o su sinónimo *familia de productos*) son:

- Grupo o familia de productos relacionados realizados por el mismo proceso y para el mismo propósito, diferenciándose sólo en el estilo, modelo o tamaño. Una línea de productos agrupa aplicaciones relacionadas en familias de aplicaciones tomando ventaja de los elementos comunes de la familia. Dado que los productos están relacionados (tienen funcionalidad o requisitos de usuario similares) hay un alto grado de aspectos comunes en una línea de productos [Sonnemann, 1995].

- Colección de productos software que recogen un conjunto de requisitos de sistema comunes, y están organizadas alrededor de una actividad específica de negocio [Cohen et al., 1995].

- Grupo de productos que comparten un conjunto de características comunes gestionadas, que satisfacen las necesidades específicas de un sector concreto del mercado [Bass et al., 1997].

- Una línea de productos consiste en una arquitectura de línea de productos y en un conjunto de elementos software reutilizables que han sido diseñados para su incorporación en la arquitectura de línea de productos. Adicionalmente, la línea de productos incluye los productos que han sido desarrollados utilizando los assets mencionados [Bosch, 2000a].

- Conjunto de productos que comparten un conjunto común de requisitos, pero que exhiben una variabilidad significativa en sus requisitos [Griss, 2000].

- Conjunto de sistemas software que comparten un conjunto de características común y gestionado, que satisface las necesidades

específicas de un segmento de mercado y que son desarrollados a partir de un conjunto central de assets de una forma establecida [SEI, 2001].

2.4.1. Beneficios y costes de una línea de productos

La principal cualidad de las líneas de productos es que potencian la reutilización estratégica. Los *assets son* componentes de una línea de productos van más allá de una mera reutilización de código. Cada producto de la línea de productos toma la ventaja del análisis, diseño, implementación, planificación, prueba, etc., realizados en cada uno de los productos desarrollados previamente en la línea de productos.

Las líneas de productos se originan inicialmente en las escuelas de negocio en la década de los ochenta, con un objetivo económico mediante el desarrollo sinérgico de productos [Knauber y Succi, 2001]. Diversos beneficios se derivan de una estrategia basada en líneas de productos, tales como la reducción de costes, el descenso del tiempo de mercado, y la mejora en la calidad. Además, se pueden esperar beneficios no puramente técnicos como resultado de la calificación de productos y de la compartición de costes.

Sin embargo, para cada posible beneficio derivado de la reutilización existe un coste asociado. En la tabla 4 [Clements et al., 1998] se presentan los elementos que afectan tanto a la línea de productos como a los nuevos productos, junto a los beneficios y los costes asociados.

Tabla 4. Beneficios y costes de las líneas de productos [Clements et al., 1998]

Asset	Beneficio	Costes
Arquitectura, especificación de la arquitectura y evaluación de la arquitectura	La arquitectura representa una inversión de diseño para la organización. Llevar esta inversión a todos los productos de la línea de productos implica que la actividad más importante del diseño esté prácticamente realizada	La arquitectura debe soportar las variaciones inherentes a la línea de productos, que imponen restricciones adicionales.
Componentes software	Los elementos derivados de la arquitectura software se comparten por los diferentes productos de la línea de productos	Los componentes deben ser diseñados de forma robusta para que puedan aplicarse en diferentes contextos, lo que complica su diseño. Con frecuencia deben diseñarse de forma más general, incluyendo los puntos de variación, sin pérdida de rendimiento
Análisis y modelado de rendimiento	Ventajas en la reutilización de análisis y modelos en el campo de los sistemas de tiempo real y sistemas distribuidos	La reutilización del análisis puede imponer restricciones sobre la movilidad de procesos entre procesadores, sobre la creación de nuevos procesos o sobre la sincronización entre procesos existentes
Pruebas, plan de pruebas, procesos de pruebas, casos de pruebas	Obtención de unidades de pruebas completas para los productos	La distinción entre la línea de productos y sus instancias significa que la solución de un problema puede tener repercusiones más allá del sistema concreto que fue corregido. Las pruebas deben considerar que todas las instancias de la línea de productos pueden verse afectadas por una modificación particular
Plan de proyecto	Los presupuestos y los calendarios pueden reutilizarse de productos previos. Además, con la experiencia previa éstos serán mejor estimados	En la práctica los beneficios derivados de la reutilización provocarán que cada producto instancia de la línea de productos tenga su propio presupuesto y calendario

Herramientas y procesos para el desarrollo de software, proceso para la realización de cambios	Herramientas del tipo de gestión de configuración, workflows se utilizan en todos los productos, ganándose en experiencia y amortizando su coste a través de la línea de productos completa	Todos estos elementos deben ser más robustos para diferenciar la gestión de la línea de productos de la gestión de un producto concreto
Personal, habilidades, formación	Debido a los elementos comunes entre las aplicaciones, se facilita la movilidad del personal entre proyectos. Su experiencia se puede aplicar a lo largo de toda la línea de productos. Su productividad crece de forma significativa	El personal debe ser formado más allá de las técnicas clásicas de ingeniería del software y de las técnicas corporativas para asegurar que entienden y pueden utilizar los assets asociados con la línea de productos. Deben crearse materiales de formación relacionados con la línea de productos. Debe producirse una transición controlada de las técnicas en ingeniería del software a las técnicas de ingeniería de dominio

3. Construcción de Componentes Core

3.1.1. Personalización de Productos

Tras analizar la línea de productos a construir y abordar la problemática de la variabilidad que supone construir una línea que sea capaz de producir una extensa gama de productos, se apreciará que existen numerosos puntos de variación, dependientes unos de otros en algunos casos y dependientes además de variantes. A su vez, algunos puntos de variación condicionan la presencia o incompatibilidad entre ellos, siendo necesario construir un árbol de dependencias y compatibilidades entre estos puntos de variación y variantes. Estas dependencias vendrán reflejadas con los operadores de relación AND, OR y XOR. A su vez, de cara a construir diferentes productos habrá que personalizar los puntos de variación a emplear.

3.1.2. Concepto de Variabilidad

La variabilidad es la habilidad de cambiar o personalizar un sistema. Introducir la variabilidad en un sistema implica facilitar cierto tipo de cambios. Es posible anticipar algunos tipos de variabilidad y construir un sistema de forma que facilite este tipo de variabilidad. Desafortunadamente, siempre hay una cierta variabilidad a la que no es posible anticiparse.

La reutilización y flexibilidad han sido las líneas conductoras por detrás del desarrollo, de tales técnicas como orientación de los objetos, frameworks orientados a objetos y líneas de producto de software.

Consecuentemente, estas técnicas permiten retrasar ciertas decisiones de diseño a un punto posterior dentro del desarrollo. Con las líneas de producto de software, la arquitectura de un sistema queda ajustada en los primeros momentos, pero los detalles de la implementación de un producto concreto se

retrasan hasta la implementación final del producto. Nos referimos a estas decisiones de diseño retrasadas como “Puntos de variación”. [Svahnberg et al., 2001]

Punto de variación: Un punto de variación es un lugar de la base de conocimiento de la línea de productos donde la variabilidad será integrada en la línea de productos. Los elementos variables como los componentes o assets, y las relaciones entre ellos se heredan de los puntos de variación de forma que indique que este elemento puede ser opcional, en caso de existir una alternativa, o los valores deben ser ajustados para solucionar la variabilidad [Laqua, 2002] .

Variante: Es una abstracción de un grupo de características relacionadas (opcionales u obligatorias). En el caso de un cliente de e-mail, una variante podría ser el editor a utilizar para redactar los mensajes.

3.1.3. Representación de la Variabilidad

Actualmente hay varias maneras de representar la variabilidad, sin que ninguna prevalezca por encima de las otras todavía.

El método más extendido hoy en día es FODA. Mediante FODA es posible representar relaciones entre componentes usando para ello diagramas de características y expresando las reglas de composición, decisiones y requisitos de diseño, todo ello a través de un catálogo de características de sistema.

La parte más significativa es, sin ningún lugar a duda, el diagrama de características. Este diagrama se expande en forma de árbol con nodos y subnodos, y cada uno de esos nodos expresa una relación sobre sus hijos de tipo AND/OR, lo cual queda expresado a través de la notación particular de FODA. El nodo raíz expresa el concepto del producto que se intenta modelizar.

Una línea que une un nodo con otro nodo expresa la relación entre ambos. Si la línea es lisa, entonces es un nodo Obligatorio (mandatory). Si por el contrario tiene un circulo en su extremo inferior, expresa un subnodo opcional. Del mismo modo, un arco que agrupa varios nodos hijos procedentes del mismo padre indica que sólo puede aparecer una especialización (nodo) de cada vez. [Robak, 2003]

Por otro lado, a través de las reglas es posible expresar otras dependencias, como la exclusión de una característica debido a la presencia de otra, o la implicación de una característica cuando aparezca otra determinada.

FODA		FORM		GP		Featu-RSEB		J. Bosch		MEANING
	Manda-tory		Manda-tory f.		Manda-tory		Compo-sed of		Compo-sition	**Mandatory** (if reachable, then f. must be chosen)
	Optio-nal		Optio-nal f.		Optio-nal		Optio-nal f.		Optio-nal f.	**Optional** (if reachable, may be chosen, or not)
	Alterna-tive		Altern a-tive f.		Alterna-tive		Vp-feature (XOR)		xor-speciali-zation	**One-of-many** choice from a group
-	-	-	-		Or-features		Vp-f use time bound (OR)		or-specia-lization	***n*-of-many** choice from a group
-	-	-	-	-	-	-	-		External feature	**External** Feature
-	-	-	-	[aFeature]	Open-feature	-	-	-	-	**Open** Feature
-	-	-	-		Prema-ture	-	-	-	-	**Premature** Feature

Ilustración 2: Diversas notaciones para representar la variabilidad.

3.1.4. Binding Time

Lo que se conoce como “binding time” o tiempo de ligadura, es el momento en el que se realiza la variabilidad de un producto software. El binding time

restringe la selección de un mecanismo. Por ejemplo, si necesita asociar una variabilidad en tiempo de ejecución, no puede ser implementada con un mecanismo limitado en tiempo de compilación [Fritsch et al. 2002a].

Los distintos tiempos de ligadura son los siguientes:

1. Programación
2. Integración
3. Ensamblaje
4. Tiempo de ejecución

En la **programación** se contempla el desarrollo de los distintos componentes core de la aplicación, así como el desarrollo del producto en si. La **integración** comprende la construcción del ejecutable en si, comprendiendo los fuentes que lo integran (código fuente, datos, ficheros de recursos, componentes de terceras empresas, ...). Durante el **ensamblaje**, el software se configura tal y como quedará al Final de la Línea. Por último, en **tiempo de ejecución**, el software se puede adaptar al hardware en cuestión durante el arranque y puesta en marcha, incluso adaptándose a posibles cambios de hardware posteriores.

3.1.5. Gestión de la Variabilidad

La mayoría del software moderno se sustenta es configuración de aspectos variables. Esta tendencia conduce a una situación donde la complejidad de manejar una gran cantidad de variabilidad se convierte en una preocupación básica que necesita ser tratada. Dos de las causas para el aumento de variabilidad son: el retraso de las decisiones del diseño al último momento económicamente factible y la transferencia de la variabilidad de la mecánica y hardware al software en sistemas empotrados.

Las líneas de productos de software toman en consideración la importancia creciente de la gestión de la variabilidad del software. Aunque el concepto de las líneas del producto de software aparece en los años 70, la incursión en la industria tiene lugar a principio al final de los años 90, de la mano de empresas como Philips, que comenzó su trabajo sobre las líneas de productos de software en 1994.

La razón de identificar la gestión de la variabilidad del software como un asunto de gran importancia es doble. Primero, junto a la comunidad de investigación de la tecnología de software, se puede mantener que la aplicación fundamental en una gama de reutilización, incluyendo frameworks orientados a objetos y tecnología de software basada en componentes y líneas de producto de software, es la gestión de la variabilidad proporcionada en el contexto de lo común. Básicamente, la reutilización de cualquier componente de software es determinada por su capacidad de apoyar la variabilidad requerida de ella. En segundo lugar, en varias organizaciones industriales, la complejidad de la gestión de la variabilidad se está convirtiendo en tal, que son necesario estudios más sistemáticos son necesarios tanto en cuanto aumentan la variabilidad. Por ejemplo, el número de los puntos de la variación para las líneas industriales de productos de software puede traducirse en miles. [Program Transformation. Software Variability Management].

4. Aplicación

4.1. Descripción del problema

La aplicación que se va a desarrollar consiste en un Sistema de Modelado y Gestión de la Variabilidad Software. A través de esta aplicación se deberán gestionar aspectos puntos de variación y sus variantes. Una vez definida esta información, el sistema nos permitirá generar esqueletos software de cada producto concreto con una documentación asociada a cada producto. Por ello esta aplicación tiene tres partes bien diferenciadas:

En la primera parte se definen las reglas de la familia de productos. Se establecerán las variantes, sus valores, puntos de variación, y características. Un punto de variación dotará al sistema de una o varias características. Del mismo modo, un punto de variación puede depender a su vez de otros puntos de variación.

En la segunda parte se definen los productos. Se pueden añadir tantos productos como se quieran y cada producto pertenecerá a una familia de productos en la cual se permiten múltiples combinaciones. Para elaborar un producto determinado a partir de una familia de productos establecida, se deberán escoger los distintos puntos de variación y variantes que lo integrarán. Esta tarea deberá poder escoger entre operadores booleanos AND, OR y XOR para elegir las partes variables de cada producto.

En la tercera parte, a partir de un proyecto ya definido, será posible generar la documentación en formato PDF del proyecto y de forma automática. La documentación nos dará información de la familia de productos utilizada, así como de las decisiones tomadas sobre el árbol de dicha familia para conseguir el producto en cuestión. Por otro lado, también podemos obtener una

estructura base en código fuente del proyecto definido. A partir de esta base, es muy sencillo y rápido completarlo.

Esta aplicación deberá tener soporte multiidioma, por el cual se pueden usar tantos idiomas como se desee sin tener que variar para ello la aplicación. Inicialmente se considerarán español e inglés. El código tipo plantilla que se genere será implementado en PHP y la interfaz y el sistema a desarrollar será una aplicación con tecnologías Web.

4.2. Análisis de Requisitos

A partir de la descripción de problema expuesto en el apartado anterior, hemos obtenido la siguiente lista de requisitos:

Tabla 5. Requisitos funcionales

Requisito	Nombre	Descripción
RF1	**Líneas de Producto (LP)**	La aplicación tendrá un sistema de mantenimiento y gestión de líneas de producto desde el que se podrán generar diferentes proyectos a partir de la línea de productos elegida. La línea de productos deberá soportar la variabilidad requerida para cada proyecto.
RF1.1	Crear nueva Línea de productos	Crea una nueva línea de productos en el sistema.
RF1.2	Abrir una Línea de productos	Abre una línea de productos de entre las disponibles.
RF1.3	Editar Línea de productos	Edita la información relativa a una línea de productos.
RF1.4	Eliminar Línea de productos	Elimina una línea de productos, junto con todos los proyectos que utilizan esta línea.
RF1.5	Cerrar Línea de productos	Cierra la línea de productos actual

Requisito	Nombre	Descripción
RF1.6	Configurar una Línea de productos	Permite configurar la línea de productos mediante la configuración de los puntos de variación existentes que integran la línea de productos seleccionada. Para ello se utilizará una estructura de árbol que permita organizar los puntos de variación y variantes en nodos.
RF1.6.1	Añadir punto de variación a la línea de productos	Con esta opción se podrá incorporar un nuevo punto de variación (de entre los existentes) a la línea de productos abierta. Para ello se aportará información como si el nuevo elemento es obligatorio u opcional, y el tipo de elección de sus nodos hijo (AND/OR/XOR).
RF1.6.2	Añadir variante a la línea de productos	Con esta opción se podrá incorporar una nueva variante (de entre las existentes en la base de datos) a la línea de productos abierta Esta variante deberá depender de un punto de variación previamente incorporado a la línea de productos. Para ello se aportará información como si el nuevo elemento es obligatorio u opcional, y el tipo de elección de sus nodos hijo (AND/OR/XOR), así como la cardinalidad
RF1.6.3	Quitar punto de variación de la línea de productos	El sistema deberá permitir eliminar un punto de variación así como todos los nodos hijo que cuelguen de él y recalcular las opciones posibles para esta línea de productos.
RF1.6.4	Quitar variante de la línea de productos	El sistema deberá permitir eliminar una variante y recalcular las opciones posibles para esta línea de productos.
RF1.6.5	Reconfigurar componente en la línea de productos	El sistema deberá permitir reconfigurar los puntos de variación presentes en el árbol, alternando estos entre puntos de variación obligatorios u opcionales, y expresando el tipo de relación con sus nodos hijos (AND/OR/XOR).
RF1.7	Visualizar Línea de productos	Muestra un esquema visual de la línea de productos activa.

Requisito	Nombre	Descripción
RF1.8	Reiniciar Línea de productos	Borra de la base de datos toda la configuración relativa a los componentes que integran la línea de productos, dejándola lista para configurarla.
RF1.9	Establecer reglas de la Línea de productos	De acuerdo a los componentes seleccionados (puntos de variación y variantes) para integrar la línea de productos, se establecen unos criterios por los cuales un componente puede o no coexistir con otro (o un valor, en el caso de ser una variante).
RF1.9.1	Relaciones de dependencia posibles	Las relaciones pueden ser de implicación o de exclusividad que se permiten son: Un Punto de variación requiere/excluye otro Punto de Variación. Un punto de variación depende/excluye de una variante. Una variante depende/excluye de otra variante.
RF1.10	Cálculo de combinaciones posibles.	Permitirá conocer cuantos productos distintos se podrán obtener operando con la variabilidad de la línea de productos. Para ello se explorará cada nodo del esquema teniendo en cuenta la relación y dependencia entre los nodos hijos (AND/OR/XOR). Mediante una función recursiva, se explorará cada nodo obteniendo un número de configuraciones distintas como resultado de la función de exploración. El valor del nodo superior (el de nivel 0) será el número de configuraciones distintas que son factibles mediante la línea de productos elegida.
RF2	Productos	La aplicación deberá poder generar distintos productos (entendiendo como producto un sistema completo funcional o una aplicación) partiendo de una línea de productos y aprovechando su variabilidad configurando los puntos de variación y variantes.

Requisito	Nombre	Descripción
RF2.1	Crear producto	El sistema deberá permitir crear nuevos productos a partir de una línea de productos que deberá estar previamente definida y almacenada en una base de datos MySQL, donde reside toda la información.
RF2.2	Abrir producto	El sistema permitirá abrir un producto cualquiera de los que existan en el repositorio para poder configurarlo, redefinirlo o cualquier otra operación.
RF2.3	Editar producto	El sistema deberá permitir editar la información relativa al producto (nombre, descripción y fecha de puesta en marcha).
RF2.4	Eliminar producto	El sistema deberá permitir eliminar toda la información relativa a un producto (nombre, descripción y fecha de puesta en marcha), así como la configuración del producto de acuerdo a la línea de productos a la que pertenece.
RF2.5	Cerrar producto	El sistema deberá permitir cerrar un producto, guardando la información relativa al mismo y decisiones tomadas sobre él hasta el momento.
RF2.6	Visualizar producto	El sistema deberá permitir ver el producto junto a su descripción y detalles. Asimismo deberá mostrar un esquema con la línea de productos en la que está basado y el camino tomado dentro de la variabilidad de la línea de productos para obtener dicho producto.
RF2.7	Verificar cumplimiento de restricciones	Una vez terminado de definir el producto a través de la línea de productos se verificarán que se cumplen las restricciones propias de la línea de productos. En caso de no satisfacerse, el sistema obligará a cambiar aquellas partes de la variabilidad que no la satisfacen. Las restricciones que deben cumplir serán del tipo implica/excluye (la presencia de un tipo de variación puede condicionar la presencia de otro o su exclusión).

Requisito	Nombre	Descripción
RF2.8	Reinicio de producto	El sistema deberá permitir reiniciar el producto borrando la configuración realizada hasta ese momento para dar la posibilidad de definirlo desde el principio.
RF2.9	Configurar producto	El sistema deberá permitir configurar el producto y adaptarlo a los requisitos dados siguiendo para ello el árbol de la línea de productos seleccionada y teniendo en cuenta las restricciones de cada nodo como obligatorio/opcional uno-de-varios/uno-o-más. Para ello, mediante una función recursiva se recorrerán los nodos del árbol conforme a las decisiones previas tomadas en la iteración anterior.
RF2.9.1	Selección de valor de una variante.	La aplicación permitirá seleccionar uno o varios valores pertenecientes a una variante. En caso de que el nodo padre tenga asignada una relación de tipo XOR, sólo se permitirá escoger un valor, y en caso de ser de tipo OR, permitirá escoger 1 o más.
RF2.9.2	Anulación de elección de valor de una variante.	El sistema permitirá eliminar selecciones anteriores de valores de variante, volviendo al estado neutro.
RF2.9.3	Selección de componentes opcionales	La aplicación permitirá incluir dentro del productos un elemento que figure en la línea de productos como opcional. Si esto sucede, en el árbol se mostrarán los nodos que dependan de este componente.
RF2.9.4	Exclusión de componentes opcionales	La aplicación permitirá excluir del producto un elemento que figure en la línea de productos como opcional. Si esto sucede, en el árbol dejarán de verse los nodos que dependan de este componente en la línea de productos.

Requisito	Nombre	Descripción
RF2.10	Cálculo de configuraciones posibles	Calculará las distintas configuraciones que dan lugar a diferentes productos a partir de la variabilidad ya configurada. Para ello se explorará cada nodo del esquema teniendo en cuenta la relación y dependencia entre los nodos hijos (AND/OR/XOR). Mediante una función recursiva, se explorará cada nodo obteniendo un número de configuraciones distintas como resultado de la función de exploración. El valor del nodo superior (el de nivel 0) será el número de configuraciones distintas que son factibles mediante la línea de productos elegida.
RF2.11	Activar punto de variación/variante	Activa la presencia de un punto de variación o variante opcional dentro del producto. Este punto de variación, al estar activado, podrá a su ver contener otros nodos, los cuales a su vez podrán ser opciones/obligatorios, y tener una relación entre ellos de tipo AND/OR/XOR.
RF3	**Puntos de variación**	El sistema tendrá una base de datos que contendrá la información de los puntos de variación presentes en el sistema.
RF3.1	Añadir punto de variación	El sistema permitirá añadir puntos de variación al repositorio.
RF3.2	Editar punto de variación	El sistema permitirá editar las características de los puntos de variación al repositorio.
RF3.3	Borrar punto de variación	El sistema permitirá eliminar puntos de variación del repositorio.
RF3.4	Consultar punto de variación	El sistema permitirá consultar la información relativa a los puntos de variación que se encuentran en el repositorio.
RF3.5	Composición de un punto de variación	Un grupo de puntos de variación definen todas las formas de las que una línea de productos puede variar. Un punto de variación es un componente o grupo de componentes que realizan una determinada función dentro del sistema.

Requisito	Nombre	Descripción
RF4	**Variantes**	El sistema tendrá un repositorio de variantes que integrarán las líneas de producto.
RF4.1	Añadir variante	Permite añadir una variante al sistema.
RF4.2	Editar variante	Permite editar un punto de variación del sistema.
RF4.3	Borrar variante	Permite borrar un punto de variación del sistema.
RF4.4	Consultar variante	Permitirá consultar la información relativa a un punto de variación del sistema.
RF4.5	Añadir valor a variante	Añade uno o más valores a una variante.
RF4.6	Eliminar valor de variante	Elimina uno o más valores de una variante.
RF5	**Cambiar idioma**	**El sistema debe permitir alternar entre distintos idiomas. La información sobre el idioma elegido residirá a nivel de sesión con variables de sesión.**
RF6	**Panel Informativo Producto**	**Mostrará la siguiente información relativa al proyecto sobre el que se está trabajando:** **nombre del producto** **Línea de productos en la que está basado** **Puntos de variación de los que consta.** **Combinaciones posibles.** **Fecha de creación.** **Fecha de modificación por última vez.**
RF7	**Panel Informativo Línea de productos**	**Mostrará la siguiente información relativa a la línea de productos sobre la que se está trabajando:** **nombre de la línea de productos** **Núm. De Productos que la implementan como base.** **Puntos de variación de los que consta.** **Combinaciones posibles.** **Fecha de creación.** **Fecha de modificación por última vez.**

Requisito	Nombre	Descripción
RF8	**Panel Informativo General**	**Mostrará la siguiente información:** **Idioma** **Num. de Líneas de productos en la base de datos** **Num. de Productos en la base de datos.** **Num. de Puntos de Variación en la base de datos.** **Num. de variantes en la base de datos.**
RF9	**Generar código**	**Una vez terminado de definir el producto, se generará código fuente correspondiente al esqueleto de la nueva aplicación. Para ello se podrá optar por un lenguaje a elegir entre PHP.**
RF10	**Generar documentación**	**A partir de las decisiones tomadas sobre la variabilidad de la línea de productos el sistema deberá poder generar documentación sobre la línea de productos y los puntos de variación y variantes utilizados y configurados para definir el producto en cuestión. Esta documentación se compondrá de una ficha relativa a cada uno de los puntos de variación y variantes intervinientes, en formato pdf mediante una llamada a una librería externa sobre PHP a la que se le pasará la información relativa al producto que reside en la base de datos.**
RF11	**Repositorio**	**El sistema contará con un almacén en una base de datos MySQL que contendrá los puntos de variación y variantes necesarios para construir las líneas de producto, y productos.**

4.3. Arquitectura necesaria.

Esta aplicación necesita la siguiente configuración para funcionar:

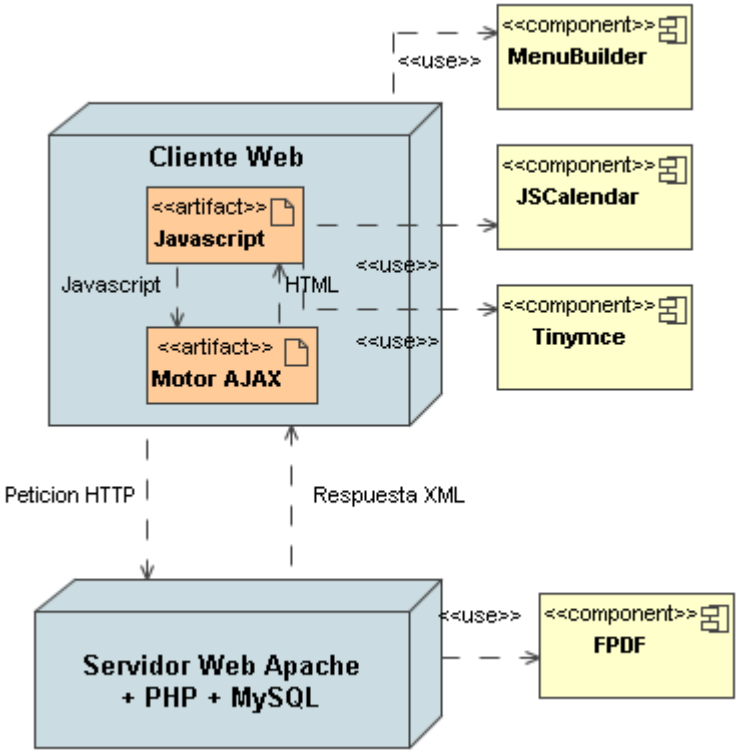


Ilustración 3: Diagrama arquitectura.

El flujo de la información es el siguiente:

El cliente hace una petición desde su navegador, el cual tiene integrada una versión de JavaScript y una implementación del objeto XMLHTTPRequest, que es el motor que impulsa la tecnología AJAX. Esta petición pasa al servidor web (Apache) que está instalado en la misma máquina en la que se encuentran PHP y MySQL. Dependiendo de la petición, podrá ser necesario tener que recurrir a bibliotecas como fPdf desde PHP. También es corriente que como consecuencia de la llamada se invoque una página PHP cuyo código tenga

instrucciones de SQL, por lo que se recuperará la información solicitada desde MySQL, y se pasará a PHP. PHP pasará una página ya traducida enteramente a HTML y Apache devolverá la página al navegador. Esta página devuelta a su vez podrá contener llamadas a ficheros JavaScript, como la aplicación que muestra el calendario, el editor HTML para campos de texto tipo textarea (TinyMCE), o el programa que genera el menú, o a hojas de estilo css, que serán recuperadas del servidor web de la misma manera e interpretadas por el cliente.

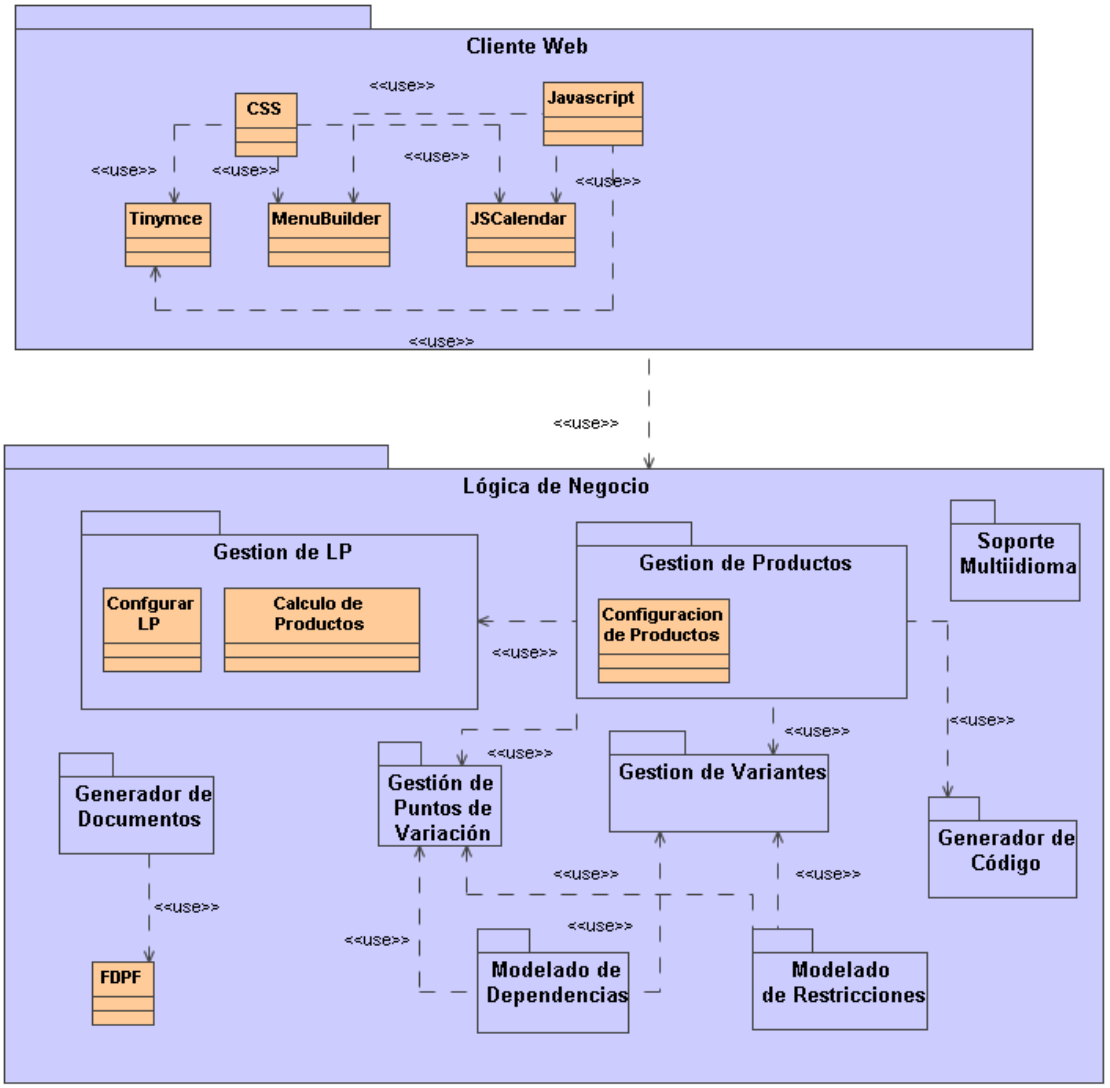


Ilustración 4: Diagrama aplicación.

Diseño de la base de datos

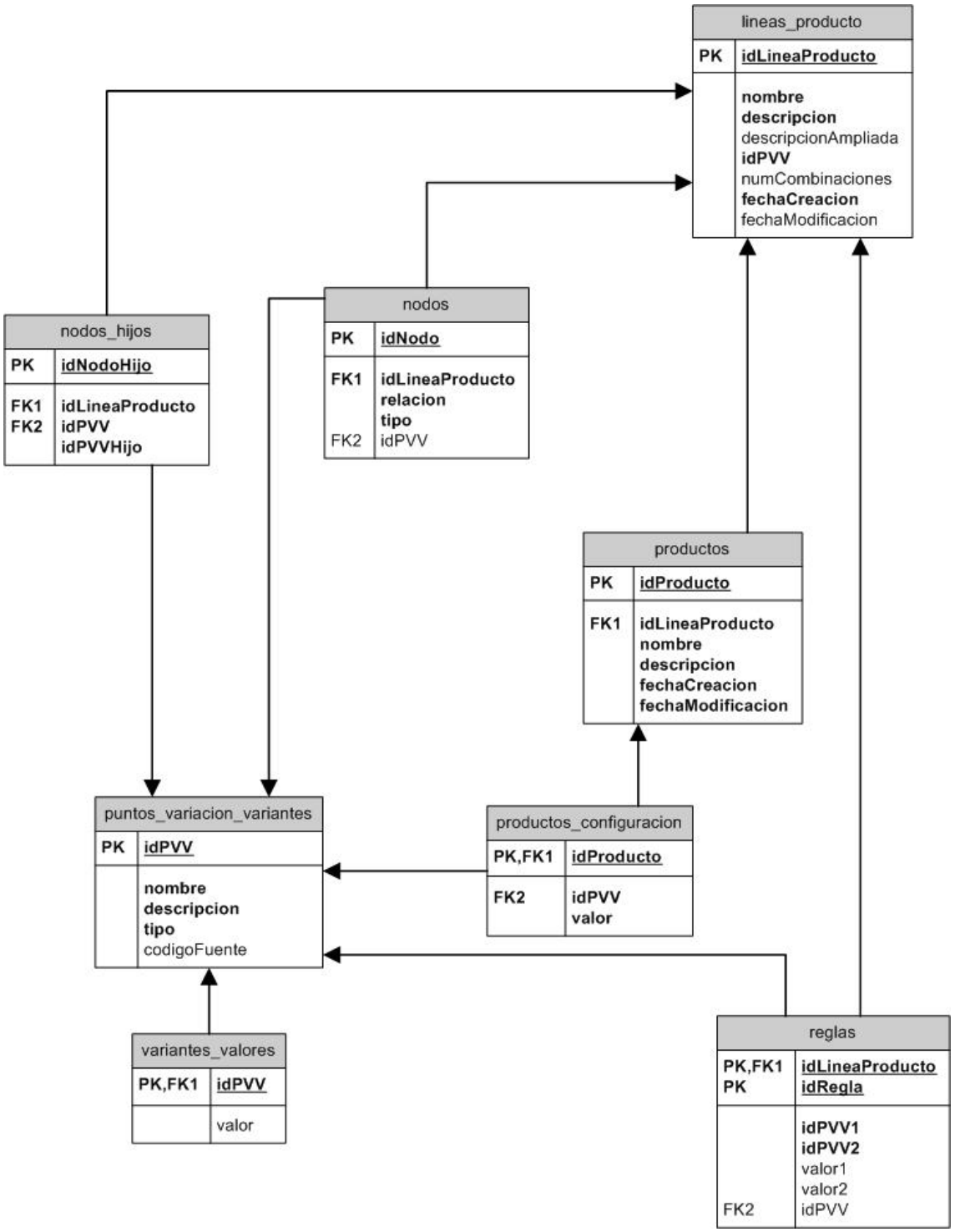


Ilustración 5: Diagrama base de datos.

4.3.1. Apache

La configuración de Apache no precisa ninguna opción especial, tan sólo debe ser configurado para que acepte las extensiones de PHP y cuando reciba una petición de este tipo, las pase al preprocesador de PHP que se encargará de generar las páginas con esta extensión y servírselas a Apache para que las devuelva al cliente.

4.3.2. CSS

Toda la aplicación está desarrollada usando hojas de estilo para optimizar el mantenimiento y la limpieza de las hojas HTML.

4.3.3. PHP

La versión elegida es PHP 5, por ser la más actual y por poder aprovechar los últimos avances de este lenguaje.

PHP debe estar configurado para funcionar con MySQL. Será capaz de conectar con MySQL a través de un socket que MySQL haya dispuesto previamente para facilitar las conexiones o bien a través del TCP/IP con un puerto (3306).

En cuanto a la configuración de php.ini, *register_globals* deben estar a *off* (register_globals=0). Esto es porque el código de esta aplicación distingue entre variables de sesión, de entorno, de parámetro y variables de programa normales..

Cada página de la aplicación abre una nueva sesión en caso de ser necesario. Por ello PHP debe configurarse para que no abra las sesiones

automáticamente (*session.auto_start=0*). El hecho de abrirlas automáticamente es algo cómodo si no se quiere llevar el control de las mismas, pero por contra, se pierde control sobre la aplicación y su desarrollo. El hecho de ponerlo a 1, aunque acelera el desarrollo inicialmente, suele provocar errores no controlados en la fase de pruebas, por lo que activar este parámetro no es recomendable.

4.3.4. MySQL

La versión utilizada es MySQL v5, que permite el uso de claves extranjeras y la utilización de forma nativa las tablas innoDB (para soportar cadenas de operaciones transaccionales) en lugar de las MyISAM que venian siendo tan frecuentes hasta la versión 4. Por otro lado, esta versión de MySQL, entre otras mejoras, permite realizar selects anidados utilizando otro select dentro de la cláusula IN.

4.3.5. PHPMyAdmin

Esta herramienta GPL sirve para administrar bases de datos remotas vía web, permitiendo la manipulación de tablas, claves, permisos, etc., así como la edición de los valores guardados en las tablas. Al ser vía web, la edición de valores es lenta, por lo que es recomendable no usarlo para este fin salvo lo estrictamente necesario (para esta función es preferible usar Toad).

4.3.6. AJAX

Introducción a AJAX

Cuando empezó el desarrollo de la web hace ya bastantes años, todo eran páginas estáticas independientes entre si, y que no compartían apenas

elementos. Al poco tiempo, surgió la idea de compartir código entre las páginas de una misma web que estuvieran relacionadas entre si. Ello se hizo mediante las primeras implementaciones de JavaScript. Por otro lado, el JavaScript era muy limitado en cuando a dinamismo. Podía hacer muchas cosas sobre una página en concreto (mostrar información en una marquesina, sacar la hora del sistema o hacer aplicaciones más o menos interactivas, pero no resolvía el gran problema de pasar datos de una página a otra. Mediante la implementación de FORMS dentro de HTML, se hacía posible enviar datos de una página a otra. Pero había que hacer algo con esos datos, pues de poco servía tenerlos en otra página si no se podía interactuar con ellos. Este papel lo realizaron durante muchos años (incluso en la actualidad) los programas de tipo Common Gateway Interface (conocidos generalmente como CGIs). CGI no era más que una interfaz, superpuesta sobre un lenguaje de programación en concreto (generalmente Perl, aunque podía ser cualquiera, como Delphi, C o Visual Basic). Sin embargo, permitía interactuar desde el lado del servidor (Server Side), y hacía cosas como grabar en una base de datos, escribir o leer ficheros de un disco del servidor o controlar el origen de las visitas. Y toda esta información sin tener que tocar la configuración del servidor web. Esto supuso una revolución en su momento, ya que posibilitaba el dinamismo que le faltaba a la web hasta ese momento. CGI se extendió de manera exponencial durante varios años, y Perl se convirtió en la referencia para hacer scripts del lado del servidor.

Primero fueron los contadores de visitas, que eran capaces de controlar el número de visitas y el número de visitantes únicos y generar pequeñas estadísticas. Luego le llegó el turno a los procesadores automáticos de formularios (forms), que a través de una programa CGI se podían recoger todas las variables enviadas desde un formulario y enviarlas a una dirección de e-mail dada. A estas aplicaciones le siguieron otras más complicadas que interactuaban con bases de datos (así aparecen los buscadores de internet). El gran problema de los CGIs es la seguridad. Los CGIs pueden ser interpretados, o programas ejecutables. Por lo tanto es preciso una configuración especial que evite su uso como herramienta de ataque. Por ello los CGIs deben estar en

un directorio aparte con permiso de ejecución, lo que merma la capacidad de estos CGIs.

De nuevo Java vuelve a ser una parte importante en la historia de la web, y aparecen los Applets. Los Applets son aplicaciones gráficas de Java que pueden ejecutarse dentro de una ventana, como un elemento html más. El problema de estas aplicaciones es que solían ser muy pesadas. Por lo que esta solución no convence y se olvida rápidamente. Macromedia había hecho lo propio con Flash y Shockwave, permitiendo también abrir un poco más la web hacia una versatilidad mayor. Inicialmente Flash y Shockwave eran algo muy parecido a los Applets en cuanto al uso que se le dio, y prácticamente no hacían nada más salvo pequeñas animaciones gráficas y usos similares. Por otro lado, y para competir con los CGIs, aparecen los Servlets de Java. La gran partida de Java es que mientras cada vez que se llamaba a un programa desde una pagina (por ejemplo el contador de visitas) con los CGIs se instanciaba una nueva copia del programa y se pasaba a memoria con el consiguiente gasto de recursos que supone, con los servlets esto no ocurría, ya que el Servlet sólo se ejecutaba una vez (el programa quedaba en memoria) y sólo se reservaba nueva memoria para las variables a usar por esa instancia, con el consiguiente ahorro de recursos.

Pero de nuevo hay una revolución en la web. Partiendo de la idea de que los CGIs permitían ejecutar código del lado del servidor e ir mostrando en el browser cliente la salida resultante, aparecen potentes lenguajes de script, tales como ASP, PHP, JSP que permiten de manera sencilla escribir scripts que son interpretados por un preprocesador que se encarga de enviar la salida resultante al navegador web.

Esto facilitaba enormemente la programación de aplicaciones para la web. S empieza a tener en cuenta a las hojas de estilo en cascada (Cascade Style Sheet o CSS) como forma de dar un aspecto similar a toda una web y ahorrando código a nivel de página (no es preciso formatear todos y cada uno de sus elementos html) y a nivel de web (la misma hoja de estilo vale para

varias páginas). Durante los años siguientes, ASP, JSP y PHP compiten como lenguajes de script del lado del servidor. Todos estos lenguajes se apoyan enormemente en las variables de sesión, que posibilita mantener en memoria un juego de variables distinto para cada visitante, algo tremendamente útil para aplicaciones como las de comercio electrónico (banca, venta electrónica, reservas, ...).

Para el intercambio de información entre aplicaciones se acuerda usar XML como método de intercambio standard. De esta forma, todas las aplicaciones podrán hablar un mismo idioma de forma que no haya que definir un protocolo nuevo para cada aplicación nueva o de cada fabricante, lo común hasta entonces. Muchos de los nuevos lenguajes empiezan a incluir bibliotecas que posibilitan el uso de XML en sus aplicaciones. Por otro lado, el XML da pie a que los lenguajes antiguos puedan usar también XML si el programador decide crear sus propias funciones de manejo de información en XML.

Pero durante todo este tiempo, perdura un problema que no tenía fácil solución y con el que todos los programadores chocaban una y otra vez:

Para interactuar con el usuario, se hacía imperativo enviar la página con el formulario completo una y otra vez. Un usuario quería darse de alta en un nuevo servicio que ofrecía cualquiera de las muchas páginas de Internet. Se le pedía rellenar un formulario, más o menos grande, y una vez rellenado, había que enviarlo para que el servidor lo procesara. Resulta muy habitual que el usuario se haya equivocado en algún campo, o que el alta no se pueda realizar porque falta información. Para prevenir esto, se puede usar JavaScript para realizar una validación simple inicial que compruebe que todos los campos obligatorios están rellenos, dejando las comprobaciones más complicadas para el script de servidor, y haciendo preciso enviar el formulario completo al servidor de cada vez. Esto es porque JavaScript inicialmente no puede realizar comprobaciones dinámicas. Es decir, si el nombre de usuario que ha escogido el visitante ya está registrado en la base de datos, JavaScript no puede hacer esa comprobación y se le deja al usuario seguir hasta que envía el formulario.

Ahora el sistema le informará de que ese nombre ya está ocupado bien porque alguien cogió ese nombre primero, o bien porque ese usuario ya se dio de alta con anterioridad y no lo recuerda. En cualquier caso, la respuesta aparecería en una nueva página, lo que hacía que se perdiera la información que no se había enviado de la anterior. Este es uno de los cientos de casos que se pueden dar y que JavaScript no era capaz de solucionar por si mismo. Y es aquí donde entra AJAX.

AJAX

AJAX viene de Asynchronous JavaScript Technology and XML. Es una tecnología representativa del fenómeno web 2.0 que da solución al problema expuesto anteriormente. Con AJAX es posible anticiparse al envío completo del formulario, enviando únicamente un trozo de información (sólo lo estrictamente necesario) y recibiendo a cambio una retroalimentación que se muestra sobre la misma pagina, dotando a la web del dinamismo que le faltaba hasta ese momento. Esto permite convertir las páginas web en auténticas aplicaciones interactivas como las clásicas aplicaciones en ventana a las que los usuarios están acostumbrados. Para usar AJAX es necesario que el cliente web tenga una implementación de JavaScript que permita instanciar el objeto **XMLHTTPRequest**.

Un caso típico es el de la selección de país, provincia, localidad. Desde hace algunos años, todos los formularios de registro de nuevo usuario suelen incluir estos campos combo. En el primero combo se elige un país, y en el segundo combo aparecen las provincias de ese país. Se selecciona una provincia y de nuevo hay una carga de datos en el tercer combo, que se llena con todas las localidades. Esto, que parece un poco “mágico”, en realidad se soluciona de una manera un poco tosca. La solución pasa por usar JavaScript y meter en arrays todos los países con todas sus provincias y todas sus localidades. Teniendo en cuenta que solo se necesita una de cada, tener que descargar varios cientos de Kb’s para luego escoger 3 elementos parece algo del todo innecesario. Lo suyo sería cargar únicamente la lista de países y no cargar la

lista de provincias hasta tener un país seleccionado. Además, tampoco es lógico enviar esa información a través de un POST y cargar una nueva página que muestre las provincias del país que viajó por el POST, teniendo que repintar la página completa de nuevo.

AJAX da una solución simple a este problema tan típico. Cuando el usuario pulsa sobre su país, se recoge su selección mediante JavaScript y se crea un nuevo objeto XMLHTTPRequest. A este objeto se le pasa el país, se le dice cual es la función JavaScript que se ejecutará cuando llegue la respuesta y se invocará una url del servidor donde hay una página en PHP (en realidad el lenguaje depende de lo que tenga instalado el servidor) que coge ese país y lo busca en la base de datos, y sacando todas las provincias pertenecientes a él. En el lado del cliente hay una función JavaScript (determinada previamente) que se ejecutará cuando llegue la información desde el servidor. Esta función, cogerá ese listado de provincias y lo cargará en el combo de provincias.

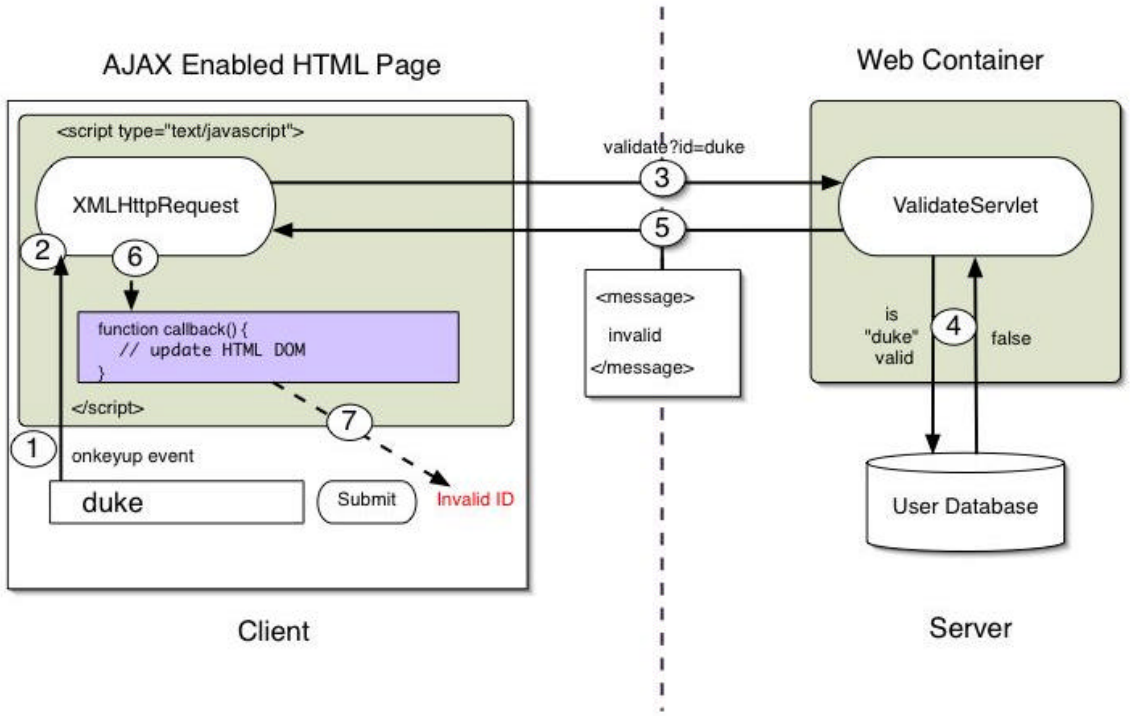


Ilustración 6: AJAX

A pesar de que AJAX está pensado para intercambiar datos con el servidor en XML, lo cierto es que puede enviar y recibir secuencias de datos con delimitadores propios, en caso de que no se quiera utilizar XML. El uso de XML implica desplegar completamente el árbol DOM, lo que implica tiempo y recursos. Para mover sólo una palabra y recibir un array de datos, quizá no sea la opción más adecuada el uso de XML. Sin embargo, su uso está más que recomendado de cara a tener una aplicación clara y legible.

A pesar de ser una tecnología bastante novedosa y relativamente reciente, sus usos son muchos, y ya está presente en una gran cantidad de páginas de la web. Algunos de sus usos más habituales son:

- Validación en tiempo real de for: Comprueban si el nombre ya está cogido, en cuyo caso muestran un mensaje de alerta o cambian el color del campo (por ejemplo Yahoo E-mail, GMail)

- Buscadores: Según el usuario escribe su búsqueda, el sistema sugiere cadenas de búsqueda posibles (p.e. Google).

- Combos entrelazados. Cuando la selección de un combo condiciona la información de otro, es muy fácil resolver el problema usando AJAX (por ejemplo Yahoo).

- Encuestas: Cuando el usuario vota en una encuesta, el marcador se actualiza automáticamente mostrando el número total de votos (por ejemplo meneame.net)

- Barras de progreso: Se puede ver el pantalla el progreso de una tarea (por ejemplo el envío de un formulario) mediante una barra (horizontal o vertical) que se va rellenando conforme se realizan las tareas en el servidor. De esta forma el usuario puede percibir que se están haciendo cosas en el servidor durante la espera, y evita que los usuarios reenvíen el formulario pensando que la petición inicial no fue atendida.

4.4. Implementación y Pruebas

4.4.1. Sesiones

Para mantener un guión y una coherencia a lo largo de una jornada de trabajo es necesario el uso de sesiones. A través de las sesiones podremos mantener en vigor una serie de variables que facilitarán el acceso a los datos que más veces se utilizan, y evitando con ello tener que preguntar esta información al usuario una y otra vez. Las variables de sesión consideradas son:

Tabla 6. Variables de sesión

Idioma	Guarda el idioma preferido por el usuario.
idProy	Guarda el identificador de proyecto con el que el usuario está trabajando.
nombreProyecto	Nombre del proyecto
descProyecto	Descripción del proyecto
fechaCre	Fecha Creación del proyecto
fechaMod	Fecha Última Modificación del proyecto
idLP	Guarda el identificador de la línea de productos con la que el usuario está trabajando
nombreLP	Nombre de la línea de productos
fechaLP	Fecha última modificación línea de productos
descLP	Descripción de la línea de productos
combiPosiblesLP	Número de productos distintos generables a partir de la línea de productos.
puntosVariacionNumero	Número de puntos de variación en el repositorio
variantesNumero	Número de variantes en el repositorio
lineasProductoNumero	Número de Líneas de productos en el repositorio
productosNumero	Número de productos existentes en el repositorio

Para evitar problemas derivados del uso de sesiones se ha optado por manejar de forma manual el comienzo de las sesiones. Por ello, al principio de cada

página se invoca la instrucción ***session_start().*** La otra opción consiste en configurar php.ini para hacer que las sesiones comiencen automáticamente (*session.auto_start = 0*) . Es cómodo, pero se pierde control sobre la aplicación.

4.4.2. Multiidioma

Esta aplicación es capaz de trabajar con varios idiomas. Para el caso que nos ocupa hemos utilizado los idiomas español e inglés.

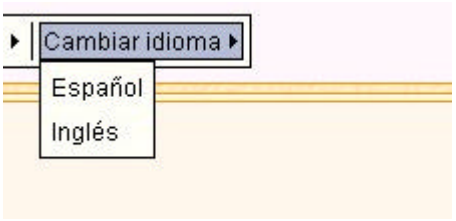


Ilustración 7: Cambio de idioma

Para conseguir esto, la aplicación trabaja únicamente con literales constantes, que están definidos en ficheros de constantes. Estos ficheros de constantes se guardan en una carpeta. Dependiendo de la variable de sesión responsable del idioma, se llamará a un fichero o a otro.

Fragmento de lang_esp.php

```
define("_ABRIR","Abrir");
define("_ACCION","Accion");
define("_ACTIVAR","Activar");
define("_ANIADIDAFUNCION","Añadida función");
define("_ANIADIDAVARIANTE","Añadida variante");
define("_ANIADIDO","añadido");
define("_ANIADIR","A&ntilde;adir");
define("_ANIADIRVALOR","Añadir valor");
define("_ANIADIRVARIANTE","La siguiente variante ha sido añadida:");
define("_BORRAR","Borrar");
define("_CANCELAR","Cancelar");
```

Fragmento de lang_eng.php

```
define("_ABRIR","Open");
define("_ACCION","Action");
define("_ACTIVAR","Activate");
define("_ANIADIDAFUNCION","Function added");
define("_ANIADIDAVARIANTE","Variant added");
define("_ANIADIDO","Added");
define("_ANIADIR","Add");
define("_ANIADIRVALOR","Add value");
define("_ANIADIRVARIANTE","Next Variant name has been added");
define("_BORRAR","Erase");
define("_CANCELAR","Cancel");
```

Con esto ya estaría el problema resuelto. Sin embargo, existe un problema y es que, al cambiar de idioma en una página cualquiera, perdemos dicha página, debiendo seleccionarla de nuevo en el menú. Para solventar este problema hemos utilizado las variables de entorno que nos proporciona el servidor, en este caso, Apache. Existe un campo, llamado ***HTTP_REFERER***, que almacena la url desde la que se llamó a la página actual. Pues bien, a través de dicha variable, es posible mandarle una orden a través de una cabecera al navegador para que redirija el flujo de salida hacia la página anterior, a través de una instrucción ***header("Location: uri")*** de forma que el cambio de idioma es inmediato, sin tiempos de espera.

```
<?
session_start();
if (isSet($_GET['lang'])) {
        $idioma=$_GET['lang'];
        if ($idioma==1) {
                $idiomanombre="Español";
        } elseif ($idioma==2) {
                $idiomanombre="English";
        } else {
                $idioma=1;
                $idiomanombre="English";
        }
        $_SESSION["idioma"]=$idioma;

} else {
        $idioma=1;
        $idiomanombre="Español";
```

```
}
$_SESSION["idioma"]=$idioma;
$_SESSION["idiomanombre"]=$idiomanombre;

@$pathVuelta=$_SERVER["HTTP_REFERER"];

if ($pathVuelta!=null) {
        header("Location: $pathVuelta");
}else{
        require("idioma.php");
        $titulo="Cambiar idioma";
        $pagina="z_changelang.php";
        require("_template.php");
}
?>
```

La opción de meter los literales en la base de datos y hacer la petición en cada página se ha desestimado desde el principio, ya que eso origina continuas peticiones de literales a la base de datos, lo que puede retardar mostrar la página. Resulta bastante más rápido realizar la carga de todos los literales desde un fichero, como en la solución prevista en esta aplicación, y de esta forma tener todos los literales disponibles a modo de *hash table*.

El multiidioma es utilizado a lo largo y ancho de toda la aplicación de forma que los comentarios insertados dentro del código fuente, o la misma documentación en PDF también puede salir en inglés en caso de ser necesario.

4.4.3. Diseño y desarrollo a través de plantillas

La construcción de la parte de presentación de la aplicación se basa en un sistema de plantillas propio, el cual permite tener una o varias plantillas intercambiables que constan de una serie de elementos gráficos, de un menú y de una o varias hojas de estilo, también intercambiables. Las distintas páginas de la web se desarrollan independientemente, únicamente conociendo los descriptores contenidos en el fichero css y utilizándolos en consecuencia, cuando sea necesario.

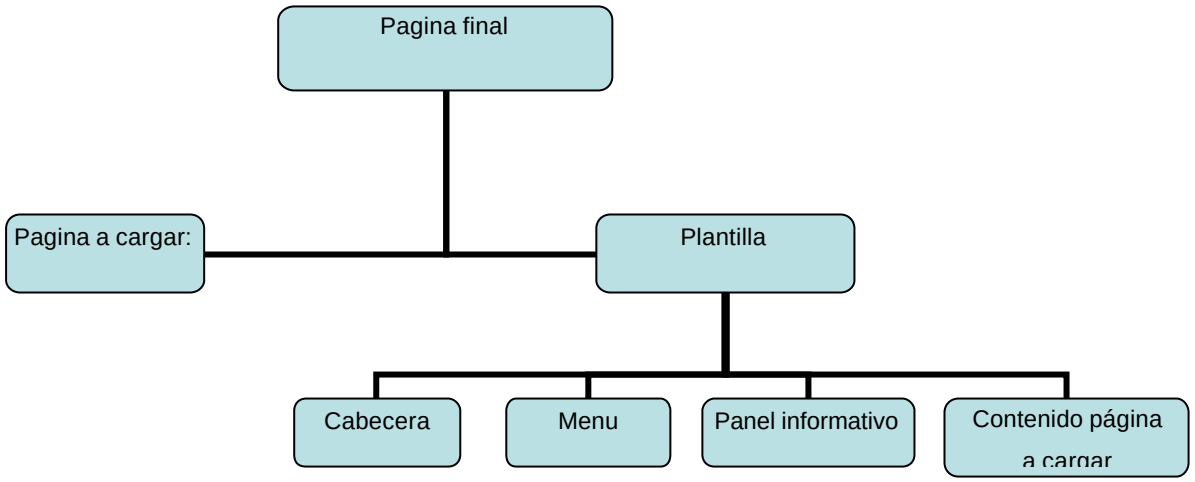


Ilustración 8: Uso de plantillas.

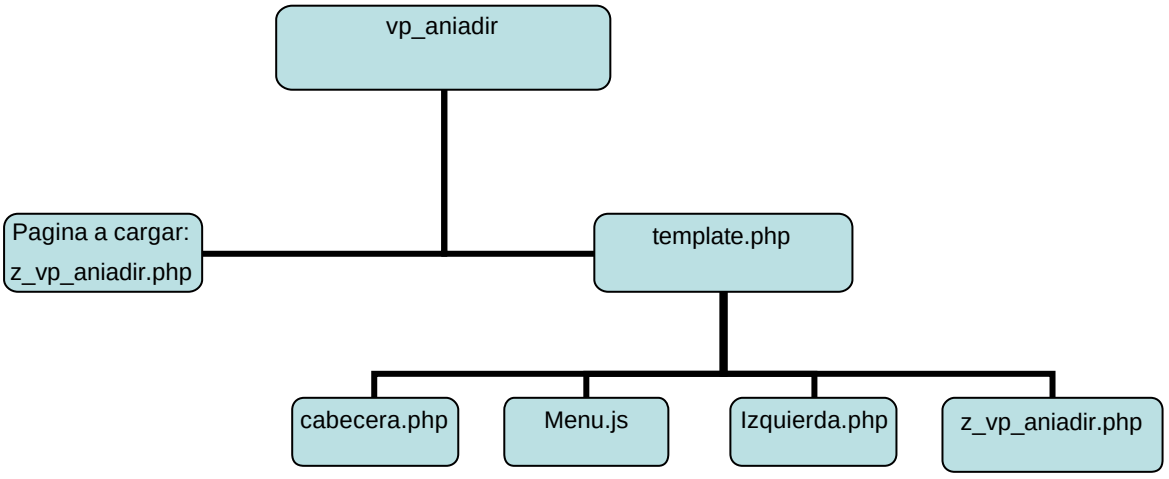


Ilustración 9: Un caso práctico de uso de plantillas

vp_aniadir.php

```
<?
session_start();
require("idioma.php");
$titulo=_VPS."-"._ANIADIR;
$pagina="z_vp_aniadir.php";
require("_template.php");
?>
```

template.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<LINK media=screen href="includes/style.css" type="text/css" rel="stylesheet">

<title><?=$titulo;?></title>
</head>
<body background="images/fondo.jpg">
<table border="0"  width="100%">
<? require(“menu.php”)?>;
<? require(“izquierda”)?>;
<? require(“pagina”)?>;
</body>
</html>
```

izquierda.php

```
<? /*AJUSTE DE VARIABLES DE SESION*/ ?>
<? /*MOSTRAR PANELES INFORMATIVOS*/ ?>
```

z_vp_aniadir.php

```
<?
        require_once("repositorio.php");
        require("db/db.php");

        $nombre=$_POST['nombre'];
        $descripcion=$_POST['descripcion'];
        $codigo=$_FILES['codigo']['name'];
```

```
        if (is_uploaded_file($_FILES['codigo']['tmp_name'])) {
                $nombreFic="ninguno.txt";
                $uploaddir = "f:\\practicas\\www\\desa\\pfc\\upload\\";
                $uploadfile = $uploaddir . basename($nombreFic);
                $a=move_uploaded_file($_FILES['codigo']['tmp_name'],$uploadfile);
                $query = "INSERT INTO vp (nombre,descripcion,varvp,codigo) VALUES
('$nombre','$descripcion','vp','".leefichero($uploadfile)."');";
        }else {
                $query = "INSERT INTO vp (nombre,descripcion,varvp) VALUES
('$nombre','$descripcion','vp');";
        }

        if (!($result = mysql_query($query, $conexion))) {
                $hayerror=1;
                $mensaje1="";
                $mensaje2= "error insert vp<br>\n";
                $mensaje2.=$query."\n";
        }else {
                $hayerror=0;
                $mensaje1="<h1>"._PUNTOVARIACIONANIADIDO.":</h1> <br><br>";
                $mensaje2=$nombre;
        }
?>
<?=$mensaje1;?>
<blockquote><span class="mensajefinal"><?=$mensaje2;?></span> </blockquote>
<br><br>
<?=_VOLVERALMENU;?>
```

A su vez también se invoca desde la plantilla aquí al fichero que tiene el menú superior con todas las opciones de la aplicación, así como los elementos gráficos de la parte superior (que al igual que el menú, también son dependientes del idioma).. En realidad hay varios ficheros JavaScript de menú, uno por cada idioma presente. Dependiendo del valor de la variable de sesión, se escogerá un fichero u otro, que se montará sobre la plantilla presente.

4.4.4. Gestión de Variantes.

Lo primero de todo, antes de empezar a construir líneas de productos, es tener preparadas las piezas que integrarán posteriormente las líneas de productos.

Por ello se definen antes de nada las variantes, que conformarán los nodos hijo del árbol que representa la línea de productos.

Para cada variante se deberán especificar una serie de valores. En caso de ser valores numéricos, se utilizarán valores cuantitativos mediante rangos ([1-100]) o valores cualitativos (velocidad alta, media, baja).

Para agilizar la asignación de valores posibles a las variantes, haciendo uso de Ajax se ha hecho posible poder introducir los valores sin cambiar de página, tal y como se haría en una aplicación de ventanas convencional:

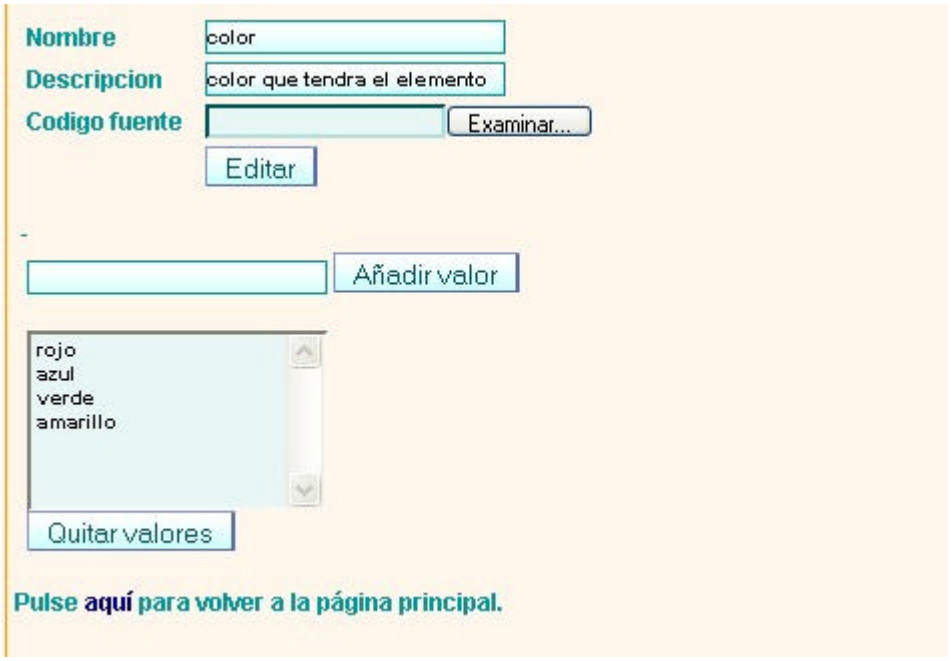


Ilustración 10: Añadiendo una variante

Por ello, desde la página de edición de las variantes se pueden controlar todos los aspectos de una variante dada, tales como el nombre, su descripción, código fuente o valores.

4.4.5. Gestión de Puntos de Variación.

A continuación debe hacerse lo mismo con los puntos de variación, ya que serán los que compongan los nodos intermedios de la línea de productos.

Del mismo modo, en la creación de un punto de variación se deben especificar su nombre, descripción y su código fuente (en caso de tenerlo).

Ilustración 11: Añadiendo un punto de variación

4.4.6. Anexar código fuente a variantes y puntos de variación.

Por otro lado, se ha utilizado la propiedad que tienen los formularios web de HTML para enviar ficheros adjuntos. Para ello es importante definir el formulario de la siguiente forma:

<form name="form1" method="post" action="vp_aniadir.php" ***enctype="multipart/form-data"****>*

La propiedad ***'enctype'*** del tag 'form' permite especificar que el contenido que se enviará en el formulario será mixto.

Ilustración 12: Anexar código al formulario

Ello permite recoger los ficheros anexos desde PHP (pueden ser imágenes, o cualquier fichero de texto o binario) y procesarlos para realizar acciones como almacenar este fichero en algún lugar del servidor, guardarlo en la base de datos o enviarlo por e-mail, por citar algunas aplicaciones típicas. En este caso, se utiliza para guardar el código fuente del componente (puntos de variación y variantes) dentro del repositorio almacenado en la base de datos.

```
$codigo=$_FILES['codigo']['name'];
if (is_uploaded_file($_FILES['codigo']['tmp_name'])) {
        $nombreFic="ninguno.txt";
        $uploaddir = "f:\\practicas\\www\\desa\\pfc\\upload\\";
        $uploadfile = $uploaddir . basename($nombreFic);
        $a=move_uploaded_file($_FILES['codigo']['tmp_name'],$uploadfile);
        $query = "INSERT INTO vp (nombre,descripcion,varvp,codigo) VALUES
('$nombre','$descripcion','vp','".leefichero($uploadfile)."');";
}else {
        $query = "INSERT INTO vp (nombre,descripcion,varvp) VALUES
('$nombre','$descripcion','vp');";
}
```

4.4.7. Gestión de Líneas de Productos

Dentro de las líneas de productos es posible realizar la gestión de los elementos de este tipo contenidos en el repositorio. Por un lado se encuentran las opciones habituales de Crear una línea de productos, abrirla, cerrarla y editarla. Tanto en **Crear** como en **Editar**, se utilizarán los campos de *Nombre de la Línea de productos*, *Descripción*, *Descripción ampliada*, *Punto de Variación Base* y *Fecha de Creación*. La siguiente tabla explicac estos campos en detalle:

Tabla 7: Campos de una línea de productos.

Nombre	Nombre que tendrá la línea de productos. Debe ser único, por lo que es recomendable comprobar que no

	existe otra línea de productos con el mismo nombre. Para ello hay un botón que permite realizar una comprobación previa (sin enviar el formulario, a través de Ajax) de que el nombre sea nuevo.
Descripción	Una breve descripción que dé una idea general de lo que puede generar esa línea de productos. Si se prefiere, es posible dar formato al texto con el editor de HTML que está incorporado.
Descripción ampliada	Descripción en profundidad de la línea de productos. Si se prefiere, es posible dar formato al texto con el editor de HTML que está incorporado.
Punto de variación base	Todas las líneas de producto deben partir desde un punto de variación que será la abstracción de los productos que se pueden generar (por ejemplo un coche). Es el nodo padre de la línea de productos y desde el que partirán todos los demás. Este nodo generalmente tiene una relación de tipo AND, ya que siempre precisa de varios componentes para poder construirse.
Fecha	Es la fecha de creación de la línea de productos. Por defecto, la fecha del día presente.

Además de estas acciones, podemos realizar otras como Configurar la línea de productos (añadiendo componentes al árbol), visualizar la línea de productos tal y como la tenemos hasta este momento, Reiniciar la línea de productos (se le borra la configuración de componentes) y Establecer Reglas que deben cumplir los productos generados a partir de esta línea.

4.4.8. Configurar la línea de productos.

La configuración de la línea de productos se lleva a cabo de forma visual y completamente intuitiva, a través de la exploración de los nodos del árbol que

aparece en pantalla. Por cada nodo es posible definir el tipo de relación que tendrá con sus hijos (AND/OR/XOR/NONE) y definir el tipo, especificando si es obligatorio (mandatory) u opcional (optional).

En cada nivel del árbol, siempre existe la posibilidad de añadir un nuevo nodo. Este nodo podrá ser un punto de variación (que a su vez podrá dar pie a otros puntos de variación o variantes) o una variante (que al seleccionarse mostrará todos los valores posibles para esta variante. Como es lógico, en los hijos de un nodo variante no pueden tener más hijos que sus propios valores, por lo que no aparece la opción de añadir punto de variación y/o variante por debajo.

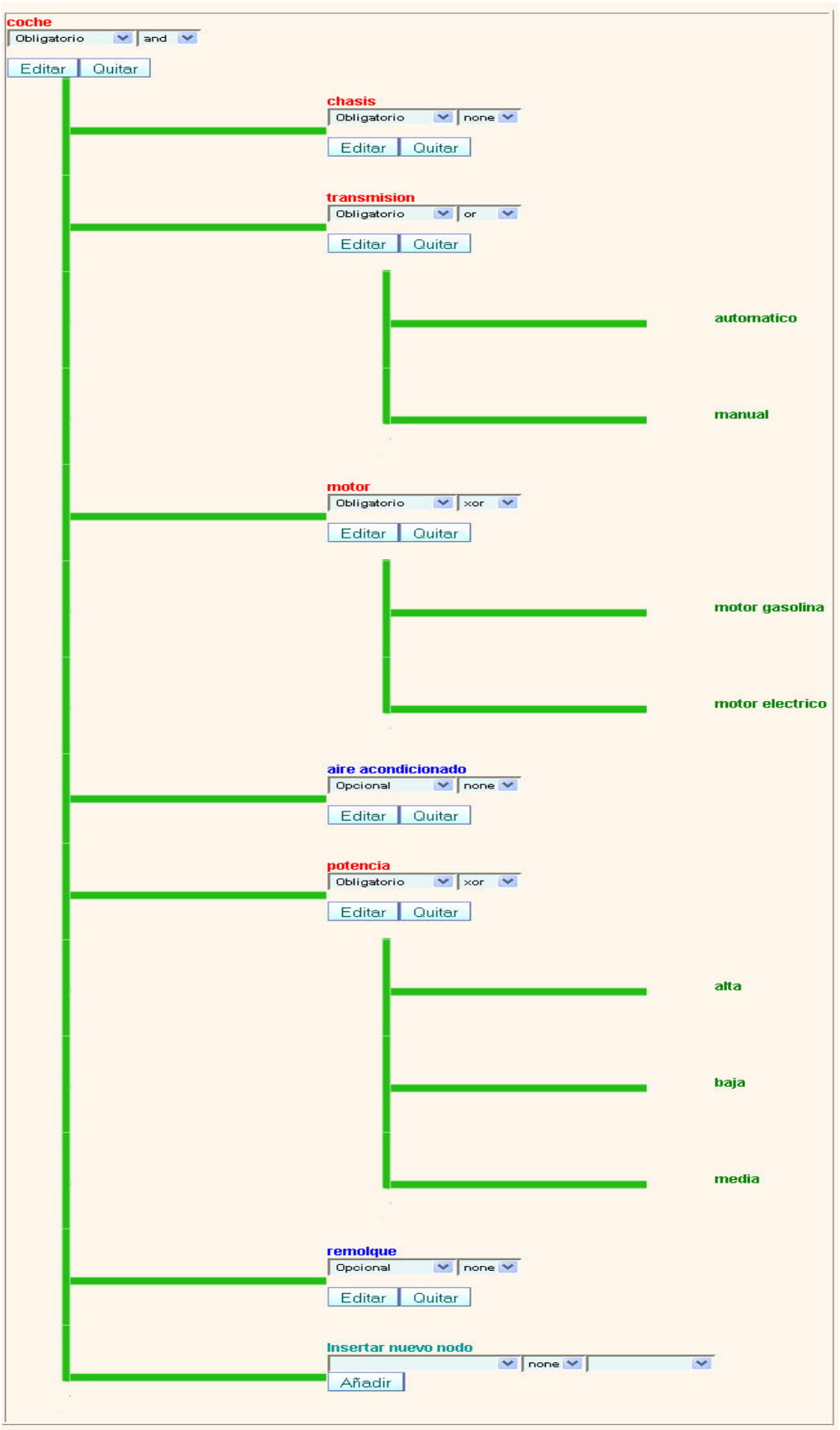


Ilustración 13: Configurando una línea de productos

Una vez configurada la línea de productos, podemos ver su aspecto final:

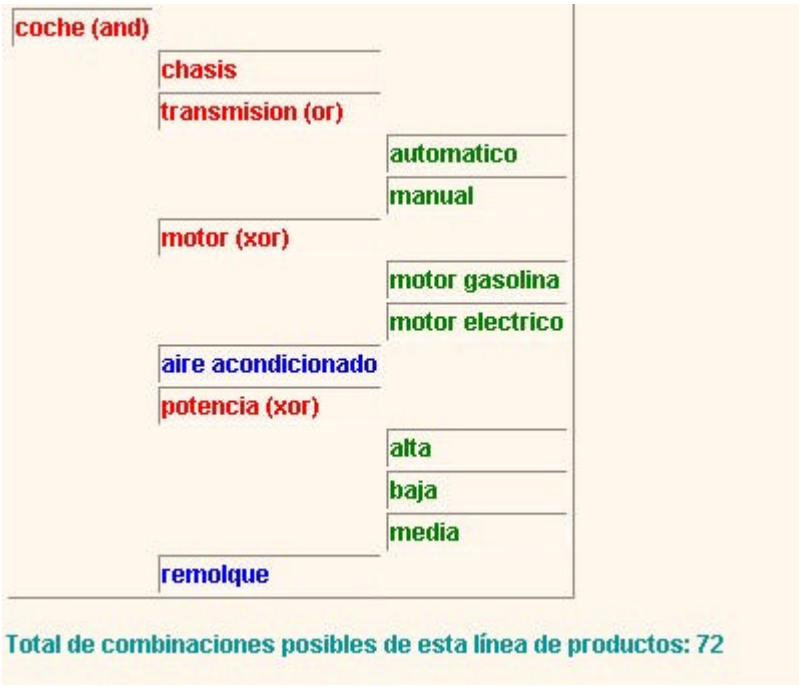


Ilustración 14: Visualizando una línea de productos

4.4.9. Cálculo del total de productos distintos generables

Ha sido necesaria la elaboración de un algoritmo que permitiera calcular cuantas combinaciones posibles distintas ofrece una línea de productos. La complejidad de este algoritmo radica en los distintos tipos de relaciones entre nodos que existen, así como de la posibilidad de que el nodo sea opcional u obligatorio.

El algoritmo funciona de la siguiente manera. Encontrándose en un nodo cualquiera, examina la relación de dicho nodo. Si se trata de una relación AND, el valor de este nodo será el producto de multiplicar la variabilidad de sus nodos entre si. Si el nodo no tuviera hijos, su variabilidad se toma como 1:

```
varNodo=varNodoHijo1 * varNodoHijo2 * … * varNodoHijon
```

En caso de ser un XOR, se sumará la variabilidad de sus nodos.

En caso de ser un OR, se sumará la variabilidad de sus nodos hijos y se le sumará 1, para incluir el caso adicional por el cual se escogerían todos los nodos (el OR puede ser que se use el hijo 1, el hijo 2, o ambos).

Cuando nos encontremos con un nodo opcional, esto contará como un punto más de cara a la variabilidad.

sacarCombinacionesNodo

```
function sacarCombiNodo($idLP,$idNodo) {
  require("db/db.php");
  $total=0;
  $cons="select relacion,tipo from nodos where idNodo=$idNodo and idLP=$idLP";
  $result = mysql_query($cons, $conexion);
  if (mysql_num_rows($result)>0) {
    list($idNodoTipoRel,$idNodoOblig)= mysql_fetch_array($result);
    if ($idNodoTipoRel=="and") {
      // SI ES UN AND, SE SUMAN LOS RESULTADOS DE SUS HIJOS
      $cons="select idNodoHijo from nodos_hijos where idNodo=$idNodo and idLP=$idLP";
      $result = mysql_query($cons, $conexion);
      $totalParcial=1;
      $calculo="1 ";
      while (list($minodo)= mysql_fetch_array($result)) {
        $miTotal=sacarCombiNodo($idLP,$minodo);
        $totalParcial=$totalParcial * $miTotal;
        $calculo=$calculo." x $minodo($miTotal)";
        }
        $total=$totalParcial;
        $nom=nombreVP($idNodo);
    } elseif ($idNodoTipoRel=="or") {
      // SI ES UN OR, SE HACE LA SUMA DE CADA UNO DE LOS NODOS +1
      if (esVariante($idNodo)) {
        $cons="select valor from variantes_valores where idVar=$idNodo";
        $result = mysql_query($cons, $conexion);
        $totalParcial=0;
        while (list($minodo)= mysql_fetch_array($result)) {
          $miTotal=1;
          $totalParcial=$totalParcial + $miTotal;
            $nom=nombreVP($minodo);
          }
          $total=$totalParcial+1;
          $nom=nombreVP($idNodo);
      }else {
        $cons="select idNodoHijo from nodos_hijos where idNodo=$idNodo and idLP=$idLP";
```

```
        $result = mysql_query($cons, $conexion);
        $totalParcial=0;
        while (list($minodo)= mysql_fetch_array($result)) {
          $miTotal=sacarCombiNodo($idLP,$minodo);
          $totalParcial=$totalParcial * $miTotal;
            $nom=nombreVP($minodo);
          }
          $total=$totalParcial+1;
          $nom=nombreVP($idNodo);
      }
    } elseif ($idNodoTipoRel=="xor") {
      // SI ES UN OR, SE HACE LA SUMA DE CADA UNO DE LOS NODOS +1
      // sacamos el raiz
      if (esVariante($idNodo)) {
        $cons="select valor from variantes_valores where idVar=$idNodo";
        $result = mysql_query($cons, $conexion);
        $totalParcial=0;
        while (list($minodo)= mysql_fetch_array($result)) {
          $miTotal=1;
          $totalParcial++;
          $nom=nombreVP($minodo);
          }
          $total=$totalParcial;
      } else {
        $cons="select idNodoHijo from nodos_hijos where idNodo=$idNodo and idLP=$idLP";
        $result = mysql_query($cons, $conexion);
        $totalParcial=0;
        while (list($minodo)= mysql_fetch_array($result)) {
          $miTotal=sacarCombiNodo($idLP,$minodo);
          $totalParcial=$totalParcial * $miTotal;
          $nom=nombreVP($minodo);
          }
          $total=$totalParcial;
      }
    } elseif ($idNodoTipoRel=="none") {
        if ($idNodoOblig=="optional") {
          $total=2;
        }else{
          $total=1;
        }
        $nom=nombreVP($idNodo);
    } else {
      echo "NO ES NINGUNO! $idNodoTipoRel<br>\n";
    }
  } else {
    $nom=nombreVP($idNodo);
    $total=1;
    }
  return($total);
}
```

4.4.10. Restricciones dentro de una línea de productos

Es necesario añadir restricciones entre componentes (puntos de variación y variantes) dentro de una línea de productos, para poder expresar situaciones como que un valor determinado de una variante es incompatible con un punto de variación determinado, o por el contrario, que la presencia de un punto de variación implica que una variante tenga un determinado valor.

Para ello existe una pantalla que permite modelizar estas situaciones.

Ilustración 15: Modelado de reglas

4.4.11. Gestión de Productos

Dentro de los proyectos o productos es posible realizar la gestión de los elementos de este tipo contenidos en el repositorio. Por un lado se encuentran las opciones habituales de 'Crear un producto', 'abrir', 'cerrar' y 'editar'. Tanto en Crear como en Editar, se utilizarán los campos de *Nombre del Producto*, *Descripción*, *Descripción ampliada*, *Línea de Productos Base* y *Fecha de Creación*. Estos campos en detalle son lo siguiente:

Tabla 7. Proyectos

Nombre	Nombre que tendrá el proyecto. Debe ser único, por lo que es recomendable comprobar que no existe otro proyecto. Para ello hay un botón que permite realizar una comprobación previa (sin enviar el formulario, a través de AJAX) de que el nombre sea nuevo.
Descripción	Una breve descripción que dé una idea general de lo que consiste ese proyecto y que tipo de variabilidad pretende modelizar.
Descripción ampliada	Descripción en profundidad del proyecto..
Línea de productos en el que se basa	Cada producto debe apoyarse en una línea de productos que será la que le dicte las pautas para saber qué puntos de variación puede usar.
Fecha	Es la fecha de creación del proyecto. Por defecto, la fecha del día presente.

Además de estas acciones, podemos realizar otras como 'Configurar el producto' (utilizando y configurando la variabilidad de la línea de productos en la que se basa), 'visualizar el producto' tal y como se ha configurado hasta este momento, 'reiniciar el producto' borrándose las opciones tomadas hasta ese momento) y 'verificar reglas' de forma que se compruebe que el producto creado no viola ninguna regla de la línea de productos en la que está basado.

4.4.12. Configuración de un producto

La configuración de un producto se lleva a cabo de forma visual y completamente intuitiva, a través de la exploración de los nodos del árbol que aparece en pantalla, basándose para ello en la línea de productos a la que pertenece el proyecto, y descendiendo por el árbol usando la información ya almacenada para el proyecto en cuestión.

Cuando aparece un nodo opcional, y este es activado, automáticamente se despliegan debajo de él aquellos nodos que dependen de este. De igual manera, cuando un nodo opcional es desactivado, desaparecen los nodos por debajo de él, y se borra de la base de datos la configuración relativa a estos nodos, si la hubiera.

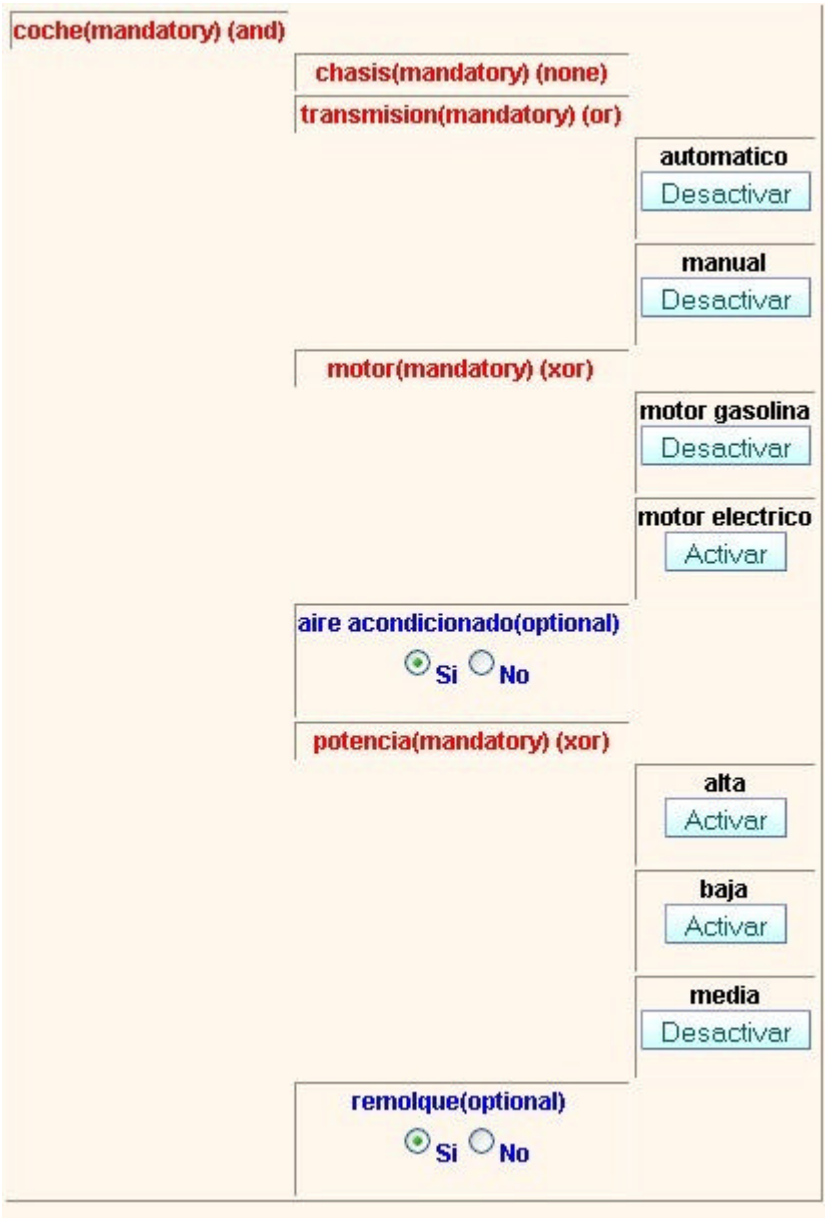


Ilustración 16: Configurando un proyecto

4.4.13. Validación de las reglas de un producto

El producto debe satisfacer las restricciones impuestas en la línea de productos. Para ello, una vez configurado, se debe usar la opción del menú para tal efecto. Si el proyecto no viola ninguna regla, aparecerá un mensaje informando de que el producto está bien ejecutado y que ya se puede generar código y documentación para él.

Ilustración 17: El proyecto cumple las reglas de la línea de productos

4.4.14. Generación de documentos en PDF

Para esta parte ha sido necesaria la utilización de la clase ***fpdf***, de licencia GPL. Esta clase permite generar documentos PDF desde PHP de manera sencilla y limpia. Esta clase permite la sobreescritura de sus métodos de forma que se adapten lo mejor posible a la aplicación. En este caso se ha hecho así, reescribiendo algunos métodos según el tipo de documento a generar.

La aplicación genera dos tipos de documentos:

- Documento sobre la línea de productos utilizada.
- Documento sobre el producto elegido y la variabilidad de la línea de productos a la que pertenece.

Ilustración 18: Generación de documentación

Pulsando sobre cada uno de los iconos se genera un documento en pdf que se abre dentro del navegador y que contiene los datos relativos bien a la línea de productos utilizada o bien al producto. En el primero de los casos se muestra información de los puntos de variación que integran la línea de productos, así como la descripción ampliada.

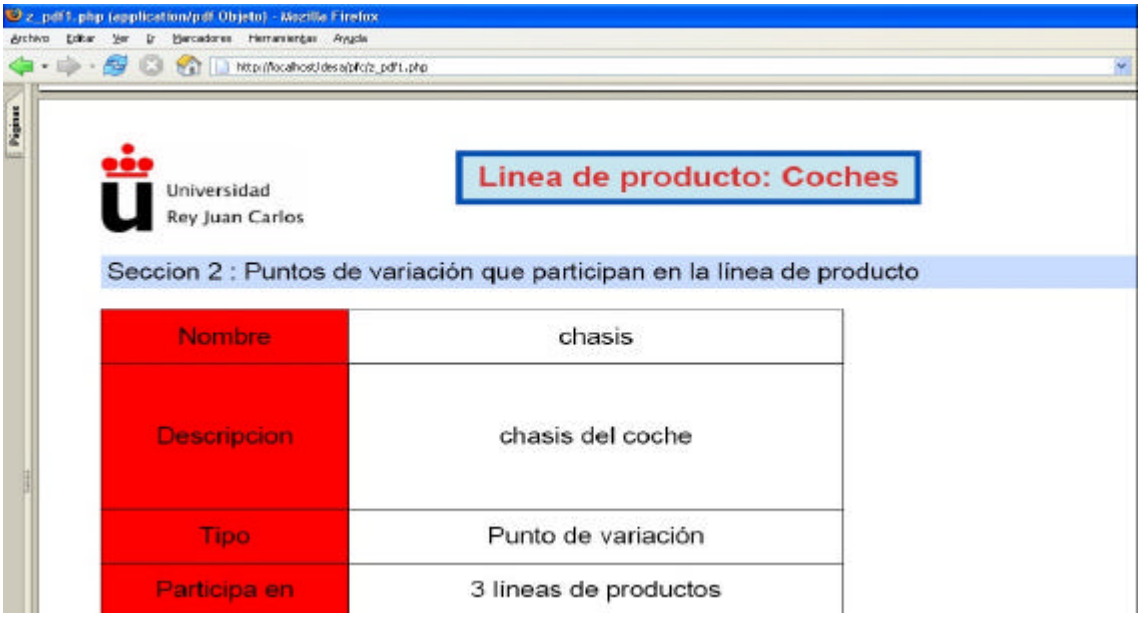


Ilustración 19: Documentación generada en PDF de una línea de productos

En el documento propio del proyecto, se puede ver la información relativa al proyecto seleccionado, así como los datos específicos del producto a generar, tales como los puntos de variación de la línea de productos que han sido utilizados para este producto, así como las variantes necesarias y su valor.

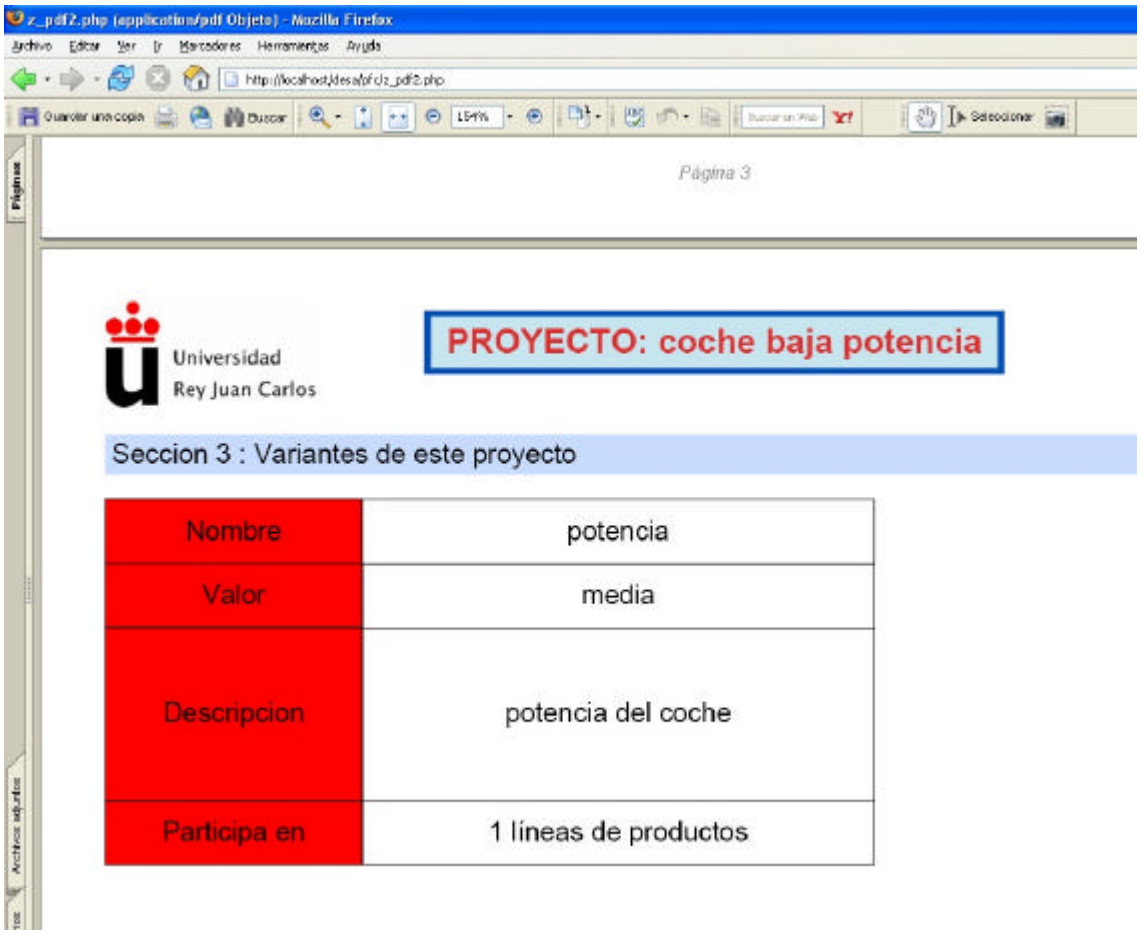


Ilustración 20: Documentación generada en PDF del proyecto

4.4.15. Generación de código fuente

Esta aplicación es capaz de generar un código fuente inicial del producto seleccionado, atendiendo a los criterios de selección realizados sobre la línea de productos, dando como resultado un producto de entre los muchos que puede generar una línea concreta.

Cuando el producto ya ha sido configurado completamente a través de las decisiones basadas en la línea de productos, nos encontramos con un proyecto que consta como mínimo de tantos ficheros como componentes intervienen en la elaboración del producto (entre puntos de variación y variantes). Los puntos de variación dan lugar a un código común que se interrelaciona con los otros componentes. Las variantes, sin embargo, dan lugar a un código fuente particular en cuyas primeras líneas se encuentran los valores que ha de tomar el componente variante para desarrollar el comportamiento deseado. Este código se especifica en la fase de diseño de los puntos de variación y variantes, al mismo tiempo que se crean.

Una vez dentro de la generación de código, dentro de la carpeta “/gen” estarán disponibles los ficheros generados para que el usuario pueda descargárselos desde la página web.

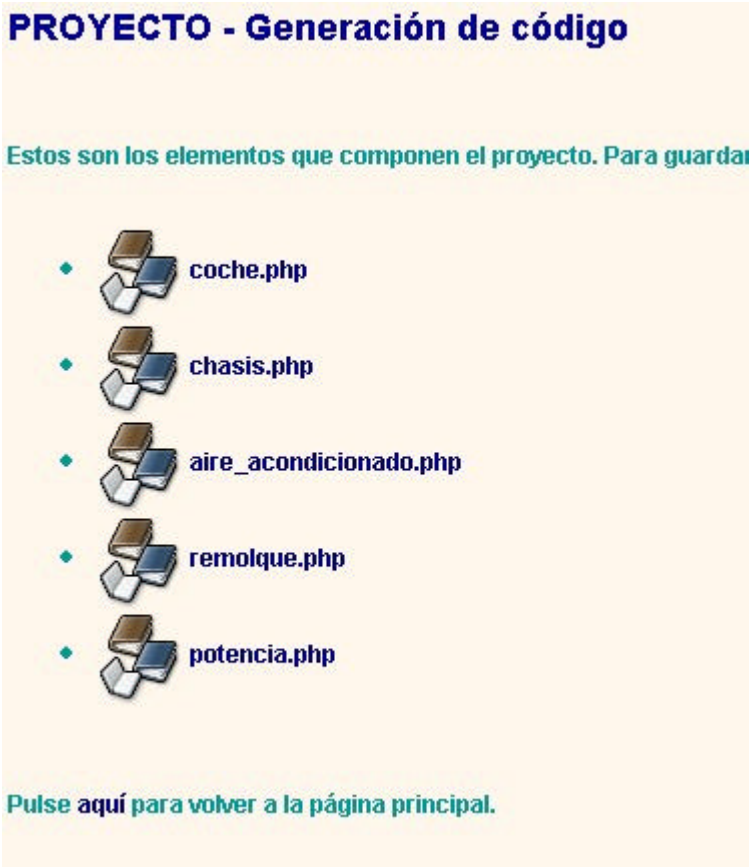
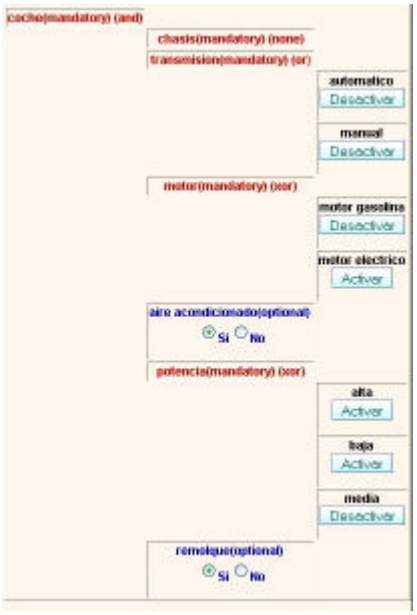
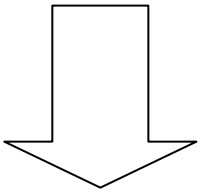


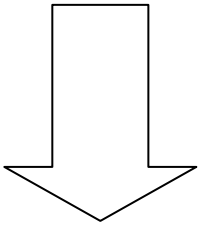
Ilustración 21: Código fuente generado.

En la ilustración 20 se puede ver como existe un fichero perteneciente al producto que se quiere generar (un coche) que a su vez se compone de otros ficheros entre puntos de variación y variantes. En un caso práctico, vamos a ver como la siguiente configuración de un coche de prestaciones normales se transforma en una aplicación.


coche(mandatory) (and)
chasis(mandatory) (none)
transmision(mandatory) (or)
automatico
Desactivar
manual
Desactivar
motor(mandatory) (xor)
motor gasolina
Desactivar
motor electrico
Activar
aire acondicionado(optional)
Si
No
potencia(mandatory) (xor)
alta
Activar
baja
Activar
media
Desactivar
remolque(optional)
Si
No




PROYECTO - Generación de código
Estos son los elementos que componen el proyecto. Para guardarlos, s
coche.php
chasis.php
aire_acondicionado.php
remolque.php
transmision.php
motor.php
potencia.php
Pulse aquí para volver a la página principal.



coche.php

```
<?
    $tipo='vp';
    require("chasis.php");
    require("aire_acondicionado.php");
    require("remolque.php");
    require("transmision.php");
    require("motor.php");
    require("potencia.php");
?>
<?


    //Codigo fuente

?>
<?
/* CODIGO FUENTE DEL COMPONENTE  */
?>
```

Ilustración 22: Flujo del proceso de generado de código fuente

Se puede observar que en las primeras líneas del código fuente generado en la ilustración 21 se hace una petición para incluir los ficheros que contienen el código de los componentes que integran el coche.

4.4.16. Apartado gráfico

Los menús y en especial, las persianas desplegables están construidas con **Sothink DHTML Menu 5.5**, una aplicación que permite desarrollar todo tipo de menús para las aplicaciones. Este tipo de ventanas están basadas en JavaScript y CSS.

Los rótulos de la aplicación están diseñados con **Xara Webstyle 4**. Se utilizan rótulos distintos para cada idioma.

Los gráficos de la aplicación web son todos ellos **PNG**. Este formato ha sido elegido por varias razones:

- No es un formato propietario.
- Soporta transparencia.

En la web, hasta hace poco, los formatos dominantes eran el JPG, que permitía ratios de compresión muy buenos y mucha calidad, y GIF, que permitía transparencias y gráficos animados (los GIF animados han sido sustituidos hoy en día por Flash en su mayor parte).

La animación de la introducción está desarrollada utilizando **Flash 5**.

La edición de las imágenes se ha hecho con **The Gimp**, un editor gráfico totalmente libre con versiones tanto para Windows como para Linux, que permite toda clase de herramientas de retoque, y también, permite manipular y convertir entre todo tipo de formatos gráficos, permitiendo PNG y las transparencias alpha, cosa que otros editores gráficos todavía no permiten.

Para el soporte multiidioma se ha desarrollado **PHP MultiLang** una aplicación específica que permite el tratamiento de literales en varios idiomas, y almacenándolos en una base de datos, la cual se puede volcar posteriormente a ficheros independientes (uno por cada idioma presente en la aplicación). Para la gestión de los distintos idiomas dentro de la aplicación, se utiliza una variable de sesión que almacena el idioma seleccionado por el usuario. Una vez que el usuario cambia de idioma, su preferencia queda almacenada y se empieza a utilizar otro fichero de literales acorde a la decisión del usuario. De esta forma, introducir un nuevo idioma es muy sencillo, pues sólo es necesario traducir todos los literales desde esta aplicación especial, lo que agiliza enormemente el trabajo, permitiendo repartirlo y permitiendo el trabajo sincronizado, y la certeza de que el programa advertirá si queda alguna clave sin traducir, emitiendo una alerta.

Por otro lado, y no menos importante, este programa es capaz de revisar las páginas PHP/HTML y extraer desde ellas los literales que encuentre, reemplazándolos con variables de constantes literales. En caso de encontrar un literal ya extraído con anterioridad, le coloca directamente en su lugar la variable que fue asignada previamente, sin necesidad de interacción por parte del usuario.

Para la captura de imágenes que se incluyen en este documento se ha utilizado una aplicación propia llamada **Captura Imágenes**, cuyo cometido es volcar las imágenes que van apareciendo en el portapapeles en un formato gráfico determinado, y con un patrón de nombre concreto (con un número incremental al final del nombre). De esta forma, con el botón Imprimir Pantalla es posible capturar rápidamente todas las imágenes necesarias en poco tiempo, sin preocuparse de darle un nombre inicial a cada imagen. En los apéndices hay más información relativa a esta aplicación.

4.4.17. HTML y Herramientas basadas en HTML/Javascript

Se han utilizado varios sistemas para dinamizar la introducción de datos a la aplicación:

En algunas partes de la aplicación se usa un calendario basado en Javascript y CSS llamado **DHTML Calendar**, de ***Dynarch***. Este calendario permite varios tipos de vistas, es completamente configurable y admite también la inclusión de nuevos temas, como es el caso. Para este proyecto, se ha creado un tema de colores especial que se ha incluido en el DHTML Calendar.

Ilustración 23: Calendario

Por último, para los campos que requieren grandes cantidades de texto, y a ser posible, debidamente formateado, como son las descripciones de los proyectos y de las líneas de producto, se ha utilizado un editor llamado TinyMCE, con licencia GPL, que permite escribir un cuadro de texto como si el usuario estuviera dentro de un procesador de texto convencional. Esta herramienta lo que hace es reescribir la forma en que el browser representa los campos ***textarea***, colocando en su lugar un editor html que permite que el usuario vea el aspecto real de lo que va escribiendo. Una vez editado el campo, cuando el formulario es enviado al servidor, lo que se envía realmente es el código html de dicho campo. Existen otras alternativas, como HTMLArea o Typepad, pero no son alternativas libres o no permiten editar libremente su código fuente.

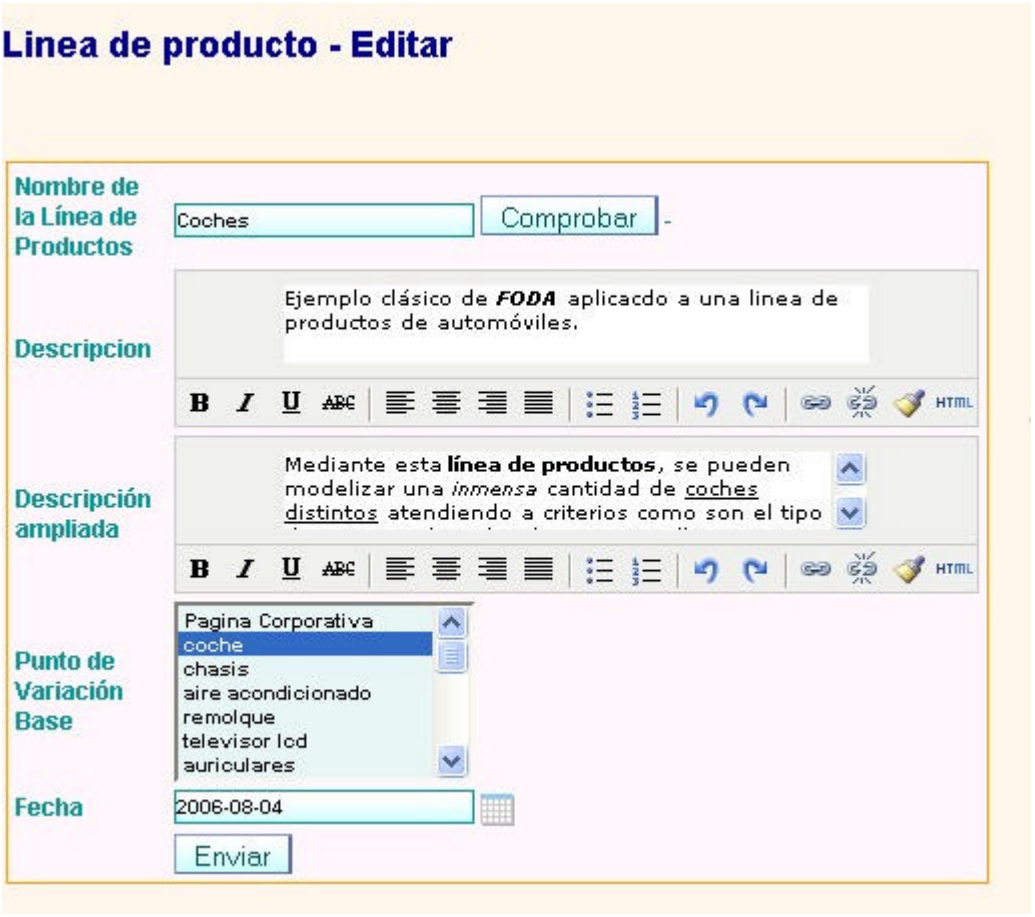


Ilustración 24: Edición de una línea de productos.

5. Conclusiones

Las conclusiones principales obtenidas de la elaboración de este proyecto son las siguientes:

Se han alcanzado los objetivos propuestos habiendo obtenido una herramienta capaz de gestionar puntos de variación en líneas de producto. Por ello, los usuarios de esta herramienta serán capaces de diseñar los productos que quieren obtener a partir de componentes reutilizables almacenados en un repositorio.

La herramienta permite introducir puntos de variación, variantes y sus valores para construir árboles tipo FODA que son utilizados para modelar y gestionar la variabilidad en un contexto de líneas de productos. Por otra parte, la herramienta no limita el tipo y número de puntos de variación existentes. Asimismo, es posible definir reglas y restricciones entre los puntos de variación y las variantes para restringir la explosión combinatoria del número de productos que, teóricamente, se podrían construir. Toda esta información se proporciona en una documentación en formato PDF que se genera de manera automática para el usuario.

De manera complementaria, la herramienta nos permite visualizar la relación de la variabilidad de los productos software para que el usuario sea capaz de trabajar con ella de forma cómoda y eficaz. Esta visualización permite desplegar la configuración de la línea de productos de forma visual e intuitiva mostrando las relaciones entre los componentes.

Por último, la herramienta resulta de gran utilidad en la construcción previa de productos software para conocer de antemano las distintas configuraciones e incompatibilidades de los productos software. La generación de un esqueleto software de forma automática resulta de gran ayuda a la hora de organizar los componentes reutilizables previamente almacenados.

Para concluir, hay que señalar que la carencia de herramientas como la que se describe en este proyecto y el incremento de complejidad de los productos software actuales, hace cada vez más necesario el uso de este tipo aplicaciones.

6. Bibliografía

[Anfurrutia et al, 2006]

Felipe I. Anfurrutia, Student Member, IEEE, Oscar Díaz, y Salvador Trujillo
"Una Aproximación de Línea de productos para la Generación de Informes de Bases de Datos"
Revista IEEE América Latina Volume: 4, Issue: 2, Date: April 2006

[Arango et. al., 1994]

Arango, G. (1994)
"Domain Analysis Methods "
W. Schäfer, et al., Software Reusability, Ellis Horwood, Hemel Hempstead, UK.

[Bass et al., 1997]

L. Bass, P. Clements, R. Kazman
"Software Architecture in Practice "
Addison-Wesley, 1997.

[Bosch, 2000a]

J. Bosch,
"Design & Use of Software Architectures: Adopting and Evolvinga Product Line Approach"
Addison-Wesley, 2000.

[Bosch et al., 2001]

J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink and K. Pohl, (2001)
"Variability Issues in Software Product Lines"
Proc. of the Fourth International Workshop on Product Family Engineering

[Buschmann et. al., 1996]

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, .Pattern
"Oriented Software Architecture: A System of Patterns"
Wiley, 1996.

[Clements et al., 1998]

Clements, Paul, Northrop, Linda M., et al.:
"A Framework for Software Product Line Practice"
Version 1.0. SEI, CMU, Sept. 1998.

[Fritsch et al. 2002a]

C. Fritsch, A. Lehn, T. Strohm: (2002)
"Binding Time"
Robert Bosch GmbH (internal slides)

[Fritsch et al. 2002b]
C. Fritsch, A. Lehn, R. Rashidi, T. Strohm:
"Variability Implementation Mechanisms – A Catalog"
Robert Bosch GmbH (internal paper)

[Gamma et al. 1995]

Gamma E., Helm R., Johnson R. and Vlissides J. (1995).
"Design Patterns—Elements of Reusable Object-Oriented Software"
Reading, Massachusetts: Addison-Wesley

[Griss, 2000]

M. L. Griss
"Implementing Product line Features with Component Reuse",
Proceedings of 6th International Conference on Software Reuse,Vienna,Austria, June 2000

[Cohen et al., 1995].

Cohen, Sholom; Friedman, Seymour; Martin, Lorraine; Solderitsch, Nancy; & Webster,Robert
"Product Line Identification for ESC-Hansom (CMU/SEI-95-SR-24)"
Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.

[Kang, 1990]

Kang, Kyo C., Sholom G. Cohen, James A Hess, William E. Novak, and A. Spencer Peterson(1990),
"Feature-Oriented Domain Analysis (FODA) Feasibility Study,"
Technical Report CMU/SEI-90-TR-21,

[Kang, 1998].

K.C. Kang,
"FORM: a feature-oriented reuse method withdomain-specific architectures"
Annals of Software Engineering, V5, pp. 354-355

[Knauber y Succi, 2001]

Peter Knauber, Giancarlo Succi
"Perspectives on Software Product Lines"
ACM SIGSOFT's SEN, January 2002

[Krzysztof et al, 2000]

Krzysztof Czarnecki and Ulrich W. Eisenecker (2000).
"Generative Programming: Methods, Tools, and Applications. "
Addison–Wesley.

[Laqua, 2002]

Roland Laqua
"Concepts for a Product Line Knowledge Base & Variability"
Proceedings of NetObjectDays 2002, Erfurt, October, 2002

[Neighbors, 1984]

J. Neighbors
"The Draco Approach to Constructing Software from Reusable Components"
IEEE Transactions on Software Engineering, 10,5,654-574,1984.

[Robak, 2003]

Silva Robak (2003)
"Feature modeling notations for system families",
International workshop - ICSE'03. Portland, USA, 2003

[SEI, 2001]

SEI Workshop on Software Architecture Representation (2001)

[Sonnemann, 1995].

Sonnemann, R. (1995)
"Exploratory Study of Software Reuse Success Factors",
Ph.D. Dissertation, George Mason University.

[Svahnberg et al., 2001]

Mikael Svahnberg, Jilles van Gurp, Jan Bosch (2001)
"On the Notion of Variability in Software Product Lines"
Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01) - Volume 00

[Svahnberg et al. 2002]

M. Svahnberg, J. van Gurp, J. Bosch:
"A Taxonomy of Variability Realization Techniques"
preprint 02/2002

Páginas web referenciadas

The Code Project. MFC and Design Patterns
http://www.codeproject.com/gen/design/mfcpatterns.asp

Carnegie Mellon Software Engineering Institute. FODA
http://www.sei.cmu.edu/domain-engineering/FODA.html

[Program Transformation. Software Variability Management]
http://www.program-transformation.org/Variability/SoftwareVariabilityManagement

[FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures]
http://selab.postech.ac.kr/publication/1998_FORM_A%20Feature-Oriented%20Reuse%20Method%20with%20Domain-Specific%20Reference%20Architectures.pdf

[Generative Programming]
http://en.wikipedia.org/wiki/Generative_programming

[GCSE working group page]
http://www-ia.tu-ilmenau.de/~czarn/generate/engl.html

[Code Generation Network]
http://www.codegeneration.net/tiki-read_article.php?articleId=64

[Sun Asynchronous JavaScript Technology and XML (AJAX) with Java]
http://java.sun.com/developer/technicalArticles/J2EE/AJAX/

7. Anexo I – Ficheros Fuente

Archivos generales

index.php: Página principal de entrada a la aplicación.
intro.html: Llamada a presentación Flash.
intro.swf: Presentación flash.
izquierda.php: Contiene los paneles informativos que aparecen en todo momento mostrando información del repositorio, de la línea de productos abierta y del proyecto abierto.
repositorio.php: Colección de 5 funciones de propósito común y 36 de propósito exclusivo utilizadas por las páginas PHP que integran la aplicación. Hay funciones tanto de manejo de ficheros como de manejo de los nodos del árbol.
template.php: Contiene la plantilla a utilizar en la aplicación.
style.css: Contiene la hoja de estilo.
menufunc.js: Funciones para construir los menús de dhtml/javascript.
tiny_mce.js: Fichero que controla la edición de los textarea en modo wysiwyg.
calendar.js: Fichero que permite construir los calendarios Dynarch.
fpdf.php: Clase que facilita la generación de ficheros pdf.
db.php: Datos de conexión a la base de datos MySQL.
images/: Directorio con las imágenes utilizadas por la aplicación.

Idioma: Ficheros responsables de realizar el cambio de idioma

changelang.php: Es el fichero que se encarga de guardar el idioma en la variable de sesión y de devolver la llamada a la página que el usurio estaba viendo.
idioma.php: se encarga de cargar la página de literales acorde al idioma
lang_esp.php: listado de constantes con sus valores en castellano.
lang_eng.php: listado de constantes con sus valores en inglés.

Ficheros máscara:

Se encargan de hacer la llamada a la plantilla pasándo el nombre del fichero que realmente contiene el código fuente.

lp_abrir.php
lp_abrir2.php
lp_cerrar.php
lp_configurar.php
lp_editar.php
lp_editar2.php
lp_editar3.php
lp_eliminar.php
lp_eliminar2.php
lp_nuevo.php
lp_nuevo2.php
lp_reglas.php
lp_reiniciar.php
lp_reiniciar2.php
lp_ver.php
proyectos.php
proyecto_abrir.php
proyecto_abrir2.php
proyecto_cerrar.php
proyecto_configurar.php
proyecto_editar.php
proyecto_editar2.php
proyecto_eliminar.php
proyecto_eliminar2.php
proyecto_nuevo.php
proyecto_nuevo2.php
proyecto_reglas.php
proyecto_reiniciar.php
proyecto_reiniciar2.php
proyecto_ver.php
vp_aniadir.php
vp_aniadir_principal.php
vp_borrar.php
vp_consultar.php
vp_consultar_principal.php
vp_editar.php
vp_editar2.php
vp_editar_principal.php
vp_eliminar_principal.php
variantes_aniadir_principal.php:
variantes_consultar_principal.php
variantes_editar_principal.php

variantes_eliminar_principal.php
variante_borrar.php
variante_consultar.php
variante_editar.php
variante_editar2.php

Variantes

z_variantes_aniadir_principal.php: Presenta la pantalla de campos para una nueva variante.
z_variante_aniadir.php: Añade una sola variante.
z_variantes_consultar_principal.php Presenta una lista de los puntos de variación para suconsulta.
z_variante_consultar.php: Muestra los datos de una variante, asi como sus valores.
z_variantes_editar_principal.php: Presenta una lista de los puntos de variación para su edición.
z_variante_editar.php: Muestra los datos de una variante, así como sus valores, los cuales se pueden insertar o eliminar gracias a AJAX.
z_variante_editar2.php: Actualiza los datos relativos a la variante.
z_variantes_eliminar_principal.php: Presenta una lista de los puntos de variación para su borrado.
z_variante_eliminar.php: Elimina la variante y sus valores.
ajax_varadd.php: Realiza la inserción de valores de la variante en el repositorio mediante AJAX.
ajax_varvaloradd.php Realiza el borrado de valores de la variante en el repositorio mediante AJAX.

Puntos de variación

z_vp_aniadir_principal.php: Presenta la pantalla de campos para un nuevo punto de variación.
z_vp_aniadir.php: Crea un nuevo punto de variación.
z_vp_eliminaraniadir_principal.php: Presenta una lista de los puntos de variación para su borrado.
z_vp_borrar.php: Borra un punto de variación
z_vp_consultar_principal.php: Presenta una lista de los puntos de variación para su consulta.
z_vp_consultar.php: Muestra los datos relativos al punto de variación.
z_vp_editar_principal.php: Presenta una lista de los puntos de variación para su edición.
z_vp_editar.php: Muestra los datos del punto de variación y permite editarlos.
z_vp_editar2.php: Actualiza los datos del puntod de variación con los datos introducidos.

Proyectos

z_proyecto_abrir.php: Presenta la lista de proyectos
z_proyecto_abrir2.php: Abre el proyecto seleccionado en la página anterior.
z_proyecto_cerrar.php: Cierra el proyecto abierto.

z_proyecto_configurar.php: Permite configurar el proyecto abierto en base a la línea de productos en la que se basa.
z_proyecto_editar.php: Presenta la lista de proyectos que hay para editar. z_proyecto_editar2.php: Edita las características principales del proyecto

z_proyecto_eliminar.php: Presenta la lista de proyectos que hay para eliminar.
z_proyecto_eliminar2.php: Elimina el proyecto seleccionado, junto con su configuración.
z_proyecto_nuevo.php: Presenta los campos necesarios para dar de alta un proyecto nuevo.
z_proyecto_nuevo2.php: Crea el proyecto.
ajax_checkproject.php: Comprueba si el nombre del proyecto ya existe en el repositorio, informando de ello mediante AJAX.
z_proyecto_reglas.php: Comprueba si el proyecto satisface las reglas impuestas por la línea de productos.
z_proyecto_reiniciar.php: Pide confirmación para hacer el reinicio.
z_proyecto_reiniciar2.php: Borra la configuración del proyecto.
z_proyecto_ver.php: Muestra un árbol con el producto generado mostrando la variabilidad elegida para el producto.

Líneas de productos

z_lp_abrir.php: Muestra una lista de las líneas de producto actuales.
z_lp_abrir2.php: Abre la línea de productos seleccionada.
z_lp_cerrar.php: Cierra la línea de productos actual.
z_lp_configurar.php: Muestra y despliega un árbol que permite añadir/editar/quitar nodos de la línea de productos actual.
z_lp_editar.php: Muestra una lista de las líneas de producto actuales y permite escoger uno para editarlo.
z_lp_editar2.php: Muestra la información de la línea de productos seleccionada.
z_lp_editar3.php: Edita la información de una línea de productos.
z_lp_eliminar.php: Muestra una lista de las líneas de producto actuales y permite escoger uno para eliminarlo.
z_lp_eliminar2.php: Elimina la línea de productos seleccionada.
z_lp_nuevo.php: Muestra los campos que hay que rellenar para crear una nueva línea de productos.
z_lp_nuevo2.php: Crea la línea de productos con la información aportada.
ajax_checklpname.php: Comprueba si el nombre de la línea de productos ya existe en el repositorio informando de ello mediante AJAX.
z_lp_reglas.php: Muestra las reglas establecidas en la línea de productos, al tiempo que permite añadir otras nuevas.
ajax_reglas.php: Se encarga de rellenar los combos de valores de acuerdo a una variante.
z_lp_reiniciar.php: Pregunta si de verdad se va a reiniciar la línea de productos actual.
z_lp_reiniciar2.php: Borra toda la configuración referente a la línea de productos.
z_lp_ver.php: Muestra un diagrama en árbol de la línea de productos actual.

Generación de código

z_generate_code_php.php: Genera los ficheros de código fuente y muestra una lista para su descarga.

Generación de Documentación en PDF

z_pdf.php: Muestra los enlaces a los documentos pdf.

z_pdf1.php: Documento pdf con la información sobre la línea de productos usada.

z_pdf2.php: Documetno pdf con la información sobre el producto generado.

8. Anexo II – Herramientas de apoyo para desarrollar la aplicación.

8.1. Captura Imágenes

Esta aplicación permite capturar de manera rápida y sencilla una batería de imágenes, que se van guardando en el disco duro con un nombre que satisface un patrón especificado, y un número incremental, de forma que en poco menos de 10 minutos es posible haber realizado 50 capturas.

Ilustración 25: Pantallla principal

En este ejemplo, se ha utilizado el patrón pfc4_%, lo que produce los ficheros pfc4_0001.bmp, pfc4_0002.bmp, pfc4_0003.bmp, … Estos ficheros pueden generarse en formatos bmp o jpg.

8.2. PHP MultiLang

Esta aplicación resulta novedosa, ya que no hay herramientas de este tipo actualmente en el mercado y automatiza y facilita gran parte del trabajo de crear un aplicación multiidioma, bien desde cero, o bien conviertiendo una aplicación ya existente.

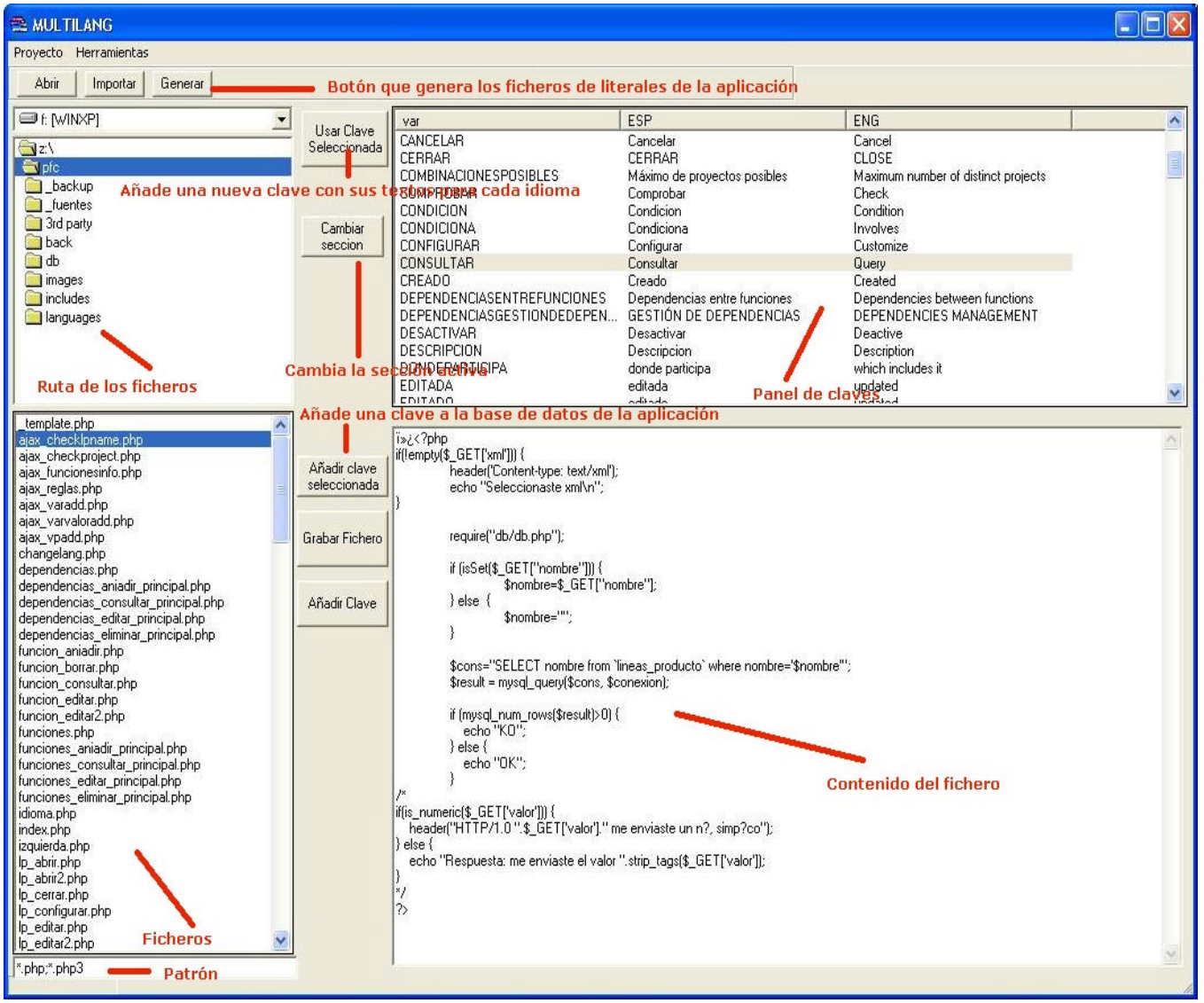


Ilustración 26: Pantalla Principal.

Las principales características de esta aplicación son:

- Creación de una base de datos de literales de una página o serie de páginas.
- Agrupación de estos literales por secciones, para permitir su organización y su rápida localización.
- Evita repetir literales (nunca habrá 2 claves con el mismo literal).
- Permite llevar un control sobre las claves
- Evita que haya inconsistencias entre los distintos idiomas (todos los idiomas tendrán exactamente las mismas claves y por consiguiente, el mismo número de estas.
- Permite insertar tantos idiomas como se quiera, incluso a partir de un proyecto previo con la base de datos llena.
- Permite encontrar fácilmente aquellos literales que no tienen traducción en un idioma determinado.
- Permite ordenar los campos por el literal o bien por la traducción en un idioma determinado.
- Cuando se introduce una nueva clave con un nuevo texto, adapta los ficheros que cumplan el patrón (por ejemplo PHP) sustituyendo en ellos el texto por la clave introducida.
- Permite editar los ficheros directamente desde la misma aplicación, por si fuera necesario realizar alguna adaptación sobre la marcha.

- Permite utilizar variables dentro de los literales, por si fuera necesario especificar un valor dentro del literal (por ejemplo “la fecha 25/17/2006 no es válida” sería “la fecha %v1% no es válida”).

- Permite importar claves y textos desde ficheros de literales previos, lo que adelanta el trabajo de introducir estos literales cuando se parte de una aplicación con uso multiidioma como puede ser phpNuke, SplattForum o phpBB, entre otros (casi todas las aplicaciones web importantes llevan ficheros de literales multiidioma).

- Genera los ficheros de literales multiidioma (uno por cada idioma).

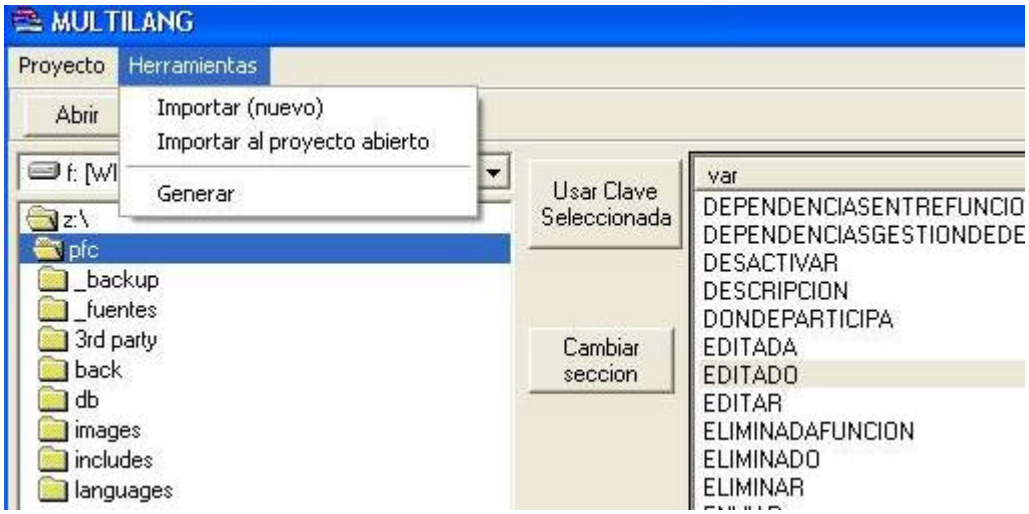


Ilustración 27: Importación.

El interfaz de esta aplicación está optimizado para trabajar rápidamente y automatizando todo el trabajo posible, de manera que el usuario sólo tenga que meter el texto de los literales nuevos.

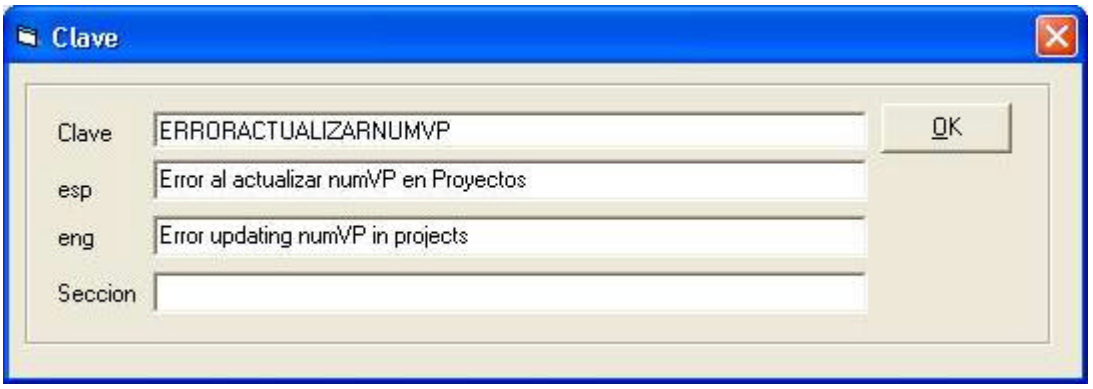


Ilustración 28: Datos de una clave.

Ilustración 29: Añadiendo una nueva clave.

El ratón juega un papel muy importante en esta aplicación. Con él se puede seleccionar texto del interior de un fichero y convertirlo a literal rápidamente.