

## TEMA 2. ANÁLISIS Y DISEÑO DE SISTEMAS WEB

---

Objetivos:

- *Conocer las peculiaridades de los sistemas software basados en Web y adaptar las metodologías clásicas de desarrollo a este entorno*
- *Ser capaz de conceptualizar en diagramas y documentos los requisitos de un SIBW*
- *Conocer distintas arquitecturas y paradigmas de diseño de sistemas Web y aplicarlas en ejercicios concretos.*

### INGENIERÍA DE REQUISITOS

---

La ingeniería de requisitos puede verse como el conjunto de actividades que lleva a la especificación de las necesidades de los usuarios y departamentos interesados en el sistema a desarrollar y las restricciones, internas o externas, que recaen sobre dicho sistema. A priori, esta definición no es muy diferente de lo que se suele aplicar a los sistemas de escritorio, pero en realidad, los sistemas software con interfaz Web tienen una serie de peculiaridades, como pueden ser:

- el interfaz está basada en hipertexto y orientada al documento
- el objetivo es mostrar información (los contenidos cambian más que la funcionalidad)
- los condicionantes de los usuarios e interesados son a menudo desconocidos de antemano
- aunque se sepa quiénes serán los interesados en el sistema, éstos no son capaces de prever cómo afectará a su modelo de negocio.

Normalmente, las aplicaciones Web forman parte de un sistema más grande, por lo que es imprescindible no sólo conocer los requisitos funcionales y no funcionales de la aplicación Web sino su contexto dentro del entorno en el que se desarrollará: interfaces con otros software, cambios en la forma de trabajar del departamento, datos necesarios por otros agentes que no utilizan el software pero sí sus resultados, etc.

#### **LEYES NO ESCRITAS (PERO REALES)**

La especificación de requisitos es un problema de comunicación.

Los interesados y el equipo de desarrollo no escriben correctamente lo que entienden como requisito.

La validación de los requisitos se produce tarde.

El usuario ve el producto final tarde

Otro aspecto a tener en cuenta y que ha de asumir antes de comenzar cualquier proyecto Web es una inherente característica a toda especificación de requisitos: la inestabilidad y variabilidad.

Como se comenta en el cuadro “Leyes no escritas (pero reales)”, la especificación de requisitos es todo un arte, dicho coloquialmente. Los factores que pueden inducir a error son numerosos, pero la mayoría están ligados a problemas de comunicación: el analista no está familiarizado con el contexto del cliente, y éste no tiene ni idea de los términos informáticos. A partir de algo tan “sencillo” se pueden liar auténticos desmadres y costar mucho tiempo y dinero en el desarrollo de cualquier producto software, y especialmente en los sistemas Web que ya vienen con la volatilidad de requisitos “de serie”.

El siguiente chiste gráfico de Dilbert lo ilustra de una forma extrema, pero que nos dará pie en clase a comentar algunos casos concretos:



### Conceptos básicos

Aunque usted ya ha cursado otras asignaturas de Ingeniería del Software, y los conceptos han debido quedar claros, no viene mal recordar los básicos de la fase de especificación de requisitos:

#### Requisitos funcionales.

Son aquellos que determinan qué debe hacer el software, qué servicios debe proporcionar y a qué datos debe reaccionar. En el caso de las aplicaciones Web, normalmente se refieren a qué elementos de entrada de datos tendrá el interfaz Web, qué funcionalidad se dispondrá, cómo se vincula una página de información con otra, etc.

De manera más concreta, los requisitos funcionales pueden referirse a:

- *Requisitos de la organización*, que comprenderán diferentes puntos de vista sobre la organización o entorno en el que se implantará la solución.
- *Requisitos del dominio de la aplicación*, que son aquellos relativos a la funcionalidad de la aplicación en sí. En el caso concreto de los datos que va a gestionar la aplicación, los requisitos se pueden referir a:
  - *Contenido de la información*
  - *Flujo de la información*
  - *Estructura de la información*
- *Requisitos de navegación*, definirán cómo pasar entre los distintos elementos de información o servicios ofrecidos por la aplicación Web.
- *Requisitos de interacción*, relativos a la interfaz de usuario y cómo los usuarios se deben desenvolver en la aplicación. La navegación es un resultado de la interacción de los usuarios con el sistema.

Como se ha comentado antes, la principal diferencia de un SIBW con respecto a un software tradicional es el impacto que tiene en el modelo de negocio de una organización. Pensemos en lo que las reservas online supusieron para las aerolíneas o los hoteles.

### *Requisitos no funcionales.*

Son aquellas restricciones que se aplican sobre la funcionalidad del sistema: plazos, estándares a utilizar, proceso de desarrollo a seguir, etc. Se suelen clasificar como:

- *Requisitos del producto.* Se refieren a las restricciones en cuanto al comportamiento del sistema: memoria utilizable, rendimiento esperado, etc. También puede incluir restricciones relativas a la resolución de pantalla, navegadores soportados, tipo de servidor, etc.
- *Requisitos organizacionales del proyecto.* Vienen impuestos por el cliente o por la empresa que desarrolla el sistema: lenguaje de programación, estándares a seguir, proceso de desarrollo, etc.
- *Requisitos externos.* Pueden ser desde normativas legales, interfaces con sistemas externos, etc.

**Ejercicio 10.** Clasifique como RF o RNF las siguientes peticiones o comentarios que nos hace el cliente sobre un software Web de gestión que vamos a desarrollar:

- “según en qué oficina tenemos ordenadores con Windows XP, Windows 7 o Windows 8, y cada trabajador elige el navegador entre IE>8, FF>3.5, y Chrome”
- “no me fío del software libre, si falla ¿quién se responsabiliza de las pérdidas o agujeros de seguridad”
- “trabajamos con varios bancos para el cobro con tarjeta,”
- “estamos en abril, y nuestra temporada fuerte es en octubre. Lo necesito ya”
- “vendría bien que se enviaran e-mails con encuestas a los clientes después de la entrega”
- “la verdad es que no sé si me acogeré al IVA de caja, creo que sí”
- “aquí tiene una factura tipo nuestra. Quiero que la que se genere sea igual”
- “necesito que se integre con mi programa de pedidos a la central”
- “se verá igual en móviles, tabletas y ordenadores ¿no?”
- “el botón de Añadir al Carro debe estar siempre bien visible”
- “aquí tiene nuestra imagen corporativa. Sígala”

### *Procesos de extracción de requisitos*

Para la extracción de requisitos hay diversas técnicas o procesos que se pueden seguir, en función del tipo de proyecto que tengamos entre manos. No es lo mismo un proyecto con un cliente que tiene claro lo que necesita, que uno en el cual se vaya a desarrollar una aplicación novedosa y sin un cliente definido.

No existe ni un proyecto software en el que los requisitos permanezcan inmutables desde el principio. Por ello es fundamental establecer criterios para identificar, controlar y seguir estos requisitos.

Es fundamental en los proyectos software, y sobre todo en los basados en plataforma Web, utilizar una metodología ágil que permita detectar cuanto antes los cambios, y minimizar las consecuencias.



En el caso de sistemas totalmente novedosos, es recomendable realizar un **estudio de viabilidad**. Es completamente temerario invertir en un desarrollo sin tener, al menos, algún especialista que conozca bien el negocio, los potenciales clientes, cómo se verán beneficiados por el producto, los canales de distribución, etc. En este caso, documentarse con bibliografía y otros casos similares también es una estrategia correcta.

En cualquiera de los casos, las **entrevistas** juegan un papel fundamental. Bien con diversos elementos del organigrama de la empresa, con empleados que usarán la aplicación, con directores comerciales, o con clientes de la empresa que requiere el desarrollo y que podrían verse beneficiados por el desarrollo.

Una técnica interesante es la **etnografía**. Consiste en estudiar sociológicamente tanto las estructuras de la empresa que requiere del nuevo desarrollo Web, como de los clientes o empleados, según el caso, que harán uso de la aplicación.

Una vez tenemos toda la información a partir de los métodos anteriores, hay que elaborarlos y redactarlos de una forma coherente y clara. Esto supone tareas de *clasificación, ordenación por prioridades, documentación y especificación* de los requisitos.

Una tarea interesante a realizar es analizar los distintos puntos de vista existentes sobre un mismo requisito, en función del afectado por el sistema. Por ejemplo, en un sistema de gestión de personal, no es la misma utilidad la que tendrá el encargado de organizar los turnos que el administrativo de RR.HH. cuya labor es elaborar las nóminas o registrar incidencias.

**Ejercicio 11.** Redacte dos puntos de vista (el del contable y el director comercial) para el siguiente requisito de una tienda online: "El sistema debe anotar todos los estados en los que se encuentre el pedido: en curso, pendiente de pago, pagado, enviado"

Una vez que están los requisitos elaborados, redactados y completos, entra en juego un aspecto muy importante: la **negociación** de los requisitos. Normalmente el cliente lo quiere todo, pero a precio limitado. Ahí entra la habilidad del director de proyecto en acordar con el cliente qué se implementará en cada fase del desarrollo.

Finalmente, es fundamental realizar una **validación** de los requisitos con el cliente. Esto puede incluso requerir una firma de un documento en el que conste que son esos, y no otros, los requisitos iniciales del proyecto.

**Ejercicio 12. (AMPLIACIÓN DE CONOCIMIENTOS)** Busque documentación fiable y complemente en sus apuntes con información sobre las metodologías **Lean** de desarrollo de proyectos.

Además del proceso de extracción de requisitos, descrito sucintamente en esta página y que desarrollaremos en la clase con más "literatura", hay ciertos aspectos que conviene recordar, aunque hayan sido vistos en asignaturas previas.

### *Dejar por escrito los requisitos*

---

Lo más habitual es que un estudiante de Ingeniería Informática considere la tarea de escribir de lo más tedioso. Pero no debería ser así. Hay que considerar que el redactar documentos es parte del trabajo ¡y muy importante!

**Ejercicio 13.** *Documente en sus apuntes la estructura básica del estándar IEEE\_29148, concretamente el apartado para especificación de requisitos. Intente aplicarlo al desarrollo de las prácticas de la asignatura.*

¿Y cómo se han de redactar estos requisitos? En lenguaje natural. ¿Por qué? Pues porque según la hipótesis de la relatividad lingüística de Sapir.Whorf, el lenguaje utilizado no sólo determina lo que dices, sino también o que puedes decir (o lo que es lo mismo, lo que se piensa). Además, hay que abandonar totalmente el lenguaje técnico, y no pensar en los detalles internos de la implementación. Esto hará que el cliente comprenda mejor lo que hemos entendido y nos pueda corregir si hemos errado.

¿Qué estructura debe tener el documento de especificación de requisitos? No es mala idea seguir el estándar del ejercicio 13, o bien una estructura como:

- Resumen
- Autor
- Descripción de los escenarios más comunes.
- Especificaciones detalladas pantalla a pantalla
- Requisitos no incluidos
- Cuestiones por resolver
- Ideas para el diseño
- Trabajos futuros
- Requisitos funcionales del usuario
- Requisitos funcionales por el dominio del problema
- Requisitos no funcionales

### *Saber extraer los requisitos*

---

Hay una expresión que usan muchos profesionales para esta etapa: “escarbar en los requisitos”, para enfatizar que lo que se realiza es extraer unos requisitos que aún ni siquiera sabe el cliente que lo son, o que muestra de una forma que puede inducir a error.

Por ejemplo:

“El sistema debe permitir elegir el periodo de amortización del préstamo” es un requisito escueto pero lo suficientemente importante como para que el ingeniero lo tenga en cuenta a la hora de diseñar el sistema. Sin embargo, lo más normal es que el cliente lo exprese en la entrevista como “los préstamos se conceden a periodos de entre 6 meses y 30 años”. Esto en realidad es una **política de empresa**, no un requisito, y podemos caer en la tentación de marcar a fuego esos márgenes en el código. Las políticas de empresa pueden cambiar, así que hay que ser muy cuidadoso con la lectura de los requisitos del cliente. Lo que habría que dejar escrito en el documento de especificación de requisitos sería algo así como “el sistema debe permitir el periodo de amortización de préstamo, que será siempre de una duración finita pero variable”.

“El usuario debe escoger el tipo de préstamo en un desplegable”. Parece un requisito, pero no lo es. Salvo que sea una obligación hacerlo así, y deberá justificarse, no es más que un ejemplo de implementación que el cliente da, pero porque él lo ve así. Hay que investigar por qué ha dicho “desplegable” y no “checkbox”.

Obviamente, esto no es una tarea fácil, pues se presentan varios problemas:

- Límites difusos del sistema. El cliente va pensando en nuevas funcionalidades conforme va explicando su sistema.
- Idiomas distintos entre el cliente y el ingeniero informático. El cliente es el experto en el dominio del problema y conoce la terminología concreta, de la misma manera que los informáticos tenemos nuestros propios esquemas mentales y terminología.
- Volatilidad. Los requisitos cambian, es así. No hay nada que hacer para evitarlo. Sólo hay que saber gestionarlo.
- Problemas no tecnológicos. Cambios de interlocutor, de políticas de la empresa, de prioridades, etc.



### Análisis del modelo de negocio, sus procesos y la audiencia del sistema

Aunque no incidiremos en demasía en este aspecto de la extracción de requisitos, es una tarea importante realizar un sosegado análisis de la influencia del sistema software, y especialmente los sistemas Web, en el modelo de negocio del cliente.

Una buena tienda online puede hacer que la empresa incremente su facturación un 1000%, o no. Todo dependerá del estudio de viabilidad y de los procesos de negocio, y la adaptación de éstos a la solución software o viceversa. Si la empresa no tiene una gestión de stock adecuada, o buenos acuerdos de mensajería, es muy posible que fracase. Por el contrario, si tiene un nicho de mercado muy específico pero con suficientes clientes potenciales, tiene gran parte del éxito garantizado.

A la hora de extraer los requisitos del sistema software hay que estudiar la audiencia potencial del producto, pues podremos descubrir jerarquías e incluso roles similares bajo nombres distintos, incluso transiciones (p.ej. un usuario que pasa de visitante anónimo a cliente por el hecho de identificarse en la Web).

### Análisis del dominio de la aplicación

El modelo del dominio de la aplicación tradicionalmente proporciona:

- *conceptos o clases*, que representan entidades relevantes en la aplicación Web
- *atributos*, que representan valores concretos o propiedades de las entidades
- *relaciones*, que representan la relación entre las distintas entidades, bien por composición, especialización/generalización o asociación.

Esta tarea de análisis de requisitos es la que mejor conocen seguramente, y es hasta comprensible realizarla en primer lugar, pues nos proporcionará un vocabulario y un panorama – estático- del dominio del problema a modelar.



El resultado del análisis del dominio de la aplicación es el ya conocido por ustedes *Diagrama de Clases Conceptuales*, por lo que no ocuparemos más tiempo ni espacio en este apartado, remitiéndole a los apuntes y bibliografía de la asignatura Fundamentos de Ingeniería del Software.

Analizar un problema lleva tiempo, pero resolver el problema equivocado lleva más tiempo aún.

En los ejercicios y prácticas, esta fase del análisis se documentará con diagramas de clase siguiendo el estándar UML.

## Análisis de la navegación y la interacción

A la hora de estudiar la interacción y navegabilidad de una aplicación Web, el enfoque es muy distinto si estamos trabajando con aplicaciones de negocio o con sistemas de propósito general.

**Ejercicio 14.** *Analice la navegabilidad de las siguientes tiendas online:*

- [www.amazon.es](http://www.amazon.es)
- [tienda.granadacf.es](http://tienda.granadacf.es)
- [www.farmakea.com](http://www.farmakea.com)
- <http://www.realmadridshop.com>

Los primeros son aquellos orientados a soportar procesos de negocio normalmente ya establecidos, por lo que la navegación está dirigida por los flujos de trabajo y los modelos de negocio de la empresa. Los sistemas de propósito general, como los portales Web o las tiendas online, no están atados a flujos predefinidos, sino que la navegabilidad se estudia y planifica para “guiar” al usuario hacia el objetivo: que compre.

Por otro lado, el análisis del espacio de interacción aborda el problema de cómo se visualizará la información en función de los distintos tipos de usuarios, y si dispondrán de diferentes acciones en cada contexto.

Ejemplo:

*Una aplicación para compra de muebles online. Su pantalla principal deberá tener:*

- *Un selector de habitaciones*
- *Un visor de plano de la habitación activa*
- *Una lista de muebles para dicha habitación*
- *Un mecanismo drag&drop que permita colocar los muebles desde la lista hasta el plano de la habitación.*
- *Poder guardar en todo momento la configuración actual*

## DISEÑO DE APLICACIONES WEB

El principal objetivo del diseño de aplicaciones Web es facilitar la comprensión de la solución que ha de desarrollarse, en lugar de comprender los usuarios y el contexto en el que ha de instalarse. Puede verse como las actividades que refinan las abstracciones identificadas durante la fase de ingeniería de requisitos con el objetivo de especificar la organización de los datos, la navegación, presentación y arquitectura de la aplicación.

Como se ha comentado anteriormente, los sistemas de información basados en Web tienen ciertas peculiaridades que los distinguen de las aplicaciones software de escritorio:

- *Mayor accesibilidad de la información y los servicios.* El acceso via WWW hace que más usuarios accedan simultáneamente a la aplicación, y posiblemente con distintos roles.
- *Interfaz orientada al documento.* La información y servicios han de transformarse en documentos de hipertexto, y la conexión y relación entre los distintos documentos requiere de un mayor nivel de abstracción
- *Variedad de tecnologías de gestión, acceso y procesamiento de datos.* Los datos pueden estar distribuidos en la Web en diversos formatos, y no ser gestionados en su totalidad por los desarrolladores de la aplicación.
- *Variedad de tecnologías y motores de visualización.* A la Web se puede acceder mediante varios motores distintos de navegación (Mozilla, Internet Explorer, Safari, Chrome, Opera, etc...) y con distintos dispositivos (móviles, PC, tabletas, etc.)
- *Arquitecturas más complejas.* La arquitectura de un sistema de información Web se asemeja más a un sistema distribuido, con balanceo de carga, distribución de los datos, arquitectura cliente-servidor de varios niveles, etc.

Vamos a centrarnos en este apartado en dar algunas pinceladas sobre metodologías de diseño de aplicaciones Web, principalmente WebML y su sucesor IFML.

### Conceptos básicos de diseño

Los principios clásicos del diseño son: abstracción, refinamiento y modularidad. La *abstracción* nos permite resolver un problema sin fijarnos en aspectos de implementación de bajo nivel, y podríamos determinar que es el primer paso para un buen diseño.

Debemos **abstraernos** de los detalles de implementación para concentrarnos en el diseño a alto nivel del flujo de trabajo, de la estructura de datos, de la navegación por el sitio o del esquema arquitectónico de la aplicación. Esto no quiere decir que sean etapas disjuntas, sino todo lo contrario. Habitualmente decisiones de diseño a nivel de estructura de datos determinan la navegación por el sitio, o funcionalidades concretas requieren de una arquitectura muy determinada, o el flujo de trabajo requiere de una interfaz concreta.

La decisión de qué priorizar en el diseño dependerá del equipo de desarrollo, si bien hay dos perfiles bien diferenciados:

- *Procesos centrados en la información,* donde lo importante son los datos que se manejan y cómo se navega por ellos, dejando para un último lugar los detalles arquitectónicos.
- *Procesos centrados en el usuario,* donde el diseño se basa en la experiencia del usuario, las actividades que va a desarrollar y qué espera encontrarse. En este punto de vista, la capa de presentación toma una especial importancia.

Conforme el proceso de diseño avanza, es necesario ir “aterrizando” e ir **refinando** los diagramas. En función del enfoque, iremos refinando los modelos de datos o los de flujo. En cualquier caso, los



datos, esquemas de navegación y propuestas de presentación se reflejan en una arquitectura que describe el funcionamiento global de la aplicación.

A la hora de refinar podemos hacer un particionamiento de la aplicación horizontal o vertical. En un particionamiento horizontal diseña por separado las páginas Web y la lógica de negocio, mientras que en un particionamiento vertical se utiliza una aproximación *top-down* donde los módulos superiores hacen las veces de controladores.

La **modularidad**, en el caso de los sistemas Web, es algo que viene casi de forma nativa en los esquemas de diseño orientados a objetos, ya que la encapsulación de información y operaciones facilita sobre manera el diseño de módulos, componentes, vistas, etc.

En cualquier caso, los objetivos del diseño en un sistema Web son claros:

- Comprender las cuestiones relacionadas con el entorno tecnológico en el que se ejecutará el software (lenguajes, tecnología de interfaz de usuario, base de datos, etc.)
- Comprender y crear un modelo que contenga los requisitos reflejados por el cliente.
- Descomponer el trabajo de implementación en tareas más pequeñas
- Detectar las principales interfaces entre distintos subsistemas.
- Definir una notación común entre los miembros del equipo de desarrollo.
- Definir una abstracción de diseño tal que la implementación no sea más que una traducción casi directa del diseño, sin cambios estructurales. Se puede usar UML o algún estándar más apropiado para Web, como IFML.

## DISEÑO DE FLUJOS DE TRABAJO

---

Para comprender la lógica de la aplicación Web, no es necesario inventar ninguna notación nueva. Los diagramas de flujo de UML y, más adelante, los de secuencia, realizan con creces la labor requerida.

## DISEÑO DE DATOS

---

El modelo de datos que especifica las estructuras que se utilizan para almacenar los datos objeto de nuestro Sistema de Información Basados en Web depende directamente de la tecnología usada para su almacenamiento.

Si se usa un sistema gestor de bases de datos relacional (MySQL, Oracle, Posgres), es evidente que el diagrama de clases UML clásico es directamente traducible a tablas y relaciones entre tablas. Por el contrario, si utilizamos una base de datos XML (como Oracle XML DB o eXist DB), hay que realizar una transformación desde el diagrama de clases de la aplicación a un árbol XML.

**Ejercicio 15.** *Refresque en sus apuntes los conocimientos adquiridos en previas asignaturas de Ingeniería del Software sobre los diagramas de flujo y los de secuencia. Inclúyalo en sus apuntes.*

**Ejercicio 16.** Complete sus apuntes con contenido sobre conversión de diagramas UML a tablas relacionales.

## DISEÑO DE LA NAVEGACIÓN

El diseño navegacional tiene que ver con la estructura de las rutas de navegación a través de la información y servicios ofrecidos por nuestro sistema de información Web. Es una actividad esencial en la ingeniería de aplicaciones Web, sobre todo cuando la Web tiene un complejo sistema de navegabilidad. Se tocan dos aspectos:

- *La estructura del sitio*, esto es, cómo los diferentes nodos o elementos de la aplicación se interconectan entre sí para componer el interfaz hipertexto
- *El comportamiento del usuario al navegar*, centrándose en las acciones y eventos que el usuario genera y que suponen cambios en la estructura hipertexto del sitio.

|   |                         |
|---|-------------------------|
| <b>Vista</b>  | <b>IFML trata sobre</b> |
| <ul style="list-style-type: none"> <li>• Composición de las vistas</li> <li>• Descripción de los elementos con los que interactúa el usuario</li> </ul>   |                         |
| <b>Controlador</b>  | <b>IFML trata sobre</b> |
| <ul style="list-style-type: none"> <li>• Especificación de los efectos de la interacción del usuario</li> <li>• Definición de eventos del sistema que el controlador debe responder.</li> </ul>       |                         |
| <b>Modelo</b>   | <b>IFML trata sobre</b> |
| <ul style="list-style-type: none"> <li>• Especificación de los elementos de datos que se muestran al usuario</li> <li>• Referencias a las acciones que son desencadenadas por los usuarios</li> </ul> |                         |

Ilustración 9: IFML en el diseño MVC

## Diseño de la estructura del sitio. IFML

En la literatura científica hay numerosas aproximaciones al diseño estructural de aplicaciones Web. En esta asignatura vamos a centrarnos en el Interaction Flow Modeling Language (IFML), que es el estándar más reciente aprobado por el OMG (Object Management Group).

Independientemente del lenguaje utilizado para dibujar el diseño (OOHDM, UWE, WebML, IFML), todos tienen una serie de componentes comunes:

- *Ítems atómicos*: elementos de información con instancias de entidades de datos. Por ejemplo: la información de un libro en amazon.com
- *Items compuestos*: estructuras compuestas de varios ítems atómicos. Por ejemplo, la información de un libro junto con la lista de sugerencias en amazon.com
- *Estructuras contextuales*: estructuras de navegación para acceder a ítems atómicos o compuestos. Por ejemplo: menús, índices, metaetiquetas...

Pasamos a explicar algo de IFML para la comprensión de este proceso de diseño de un Sistema de Información Basado en Web.

IFML en realidad es un lenguaje de modelado independiente de plataforma, esto es, lo mismo sirve para diseñar el interfaz de un sistema Web que de una aplicación de escritorio, para tabletas o para teléfonos móviles. El foco de la descripción que se hace del sistema se pone en la estructura y comportamiento de la aplicación según lo ve el usuario final, mientras que la estructura y comportamientos de la lógica del programa, de los componentes de datos y de negocio, se limita a tan sólo aquellos aspectos en los que interviene el usuario final.

Podríamos decir que en el esquema Modelo-Vista-Controlador (MVC), IFML se centra en la Vista, aunque describe cómo la vista referencia o depende del modelo y los controladores de la aplicación, tal y como se muestra en la ilustración 9.

Está claro que con un ámbito tan exiguo, IFML se queda corto. Sin embargo, UML dispone de herramientas para cubrir esas carencias con creces:

- El funcionamiento interno de las acciones desencadenadas por la interacción del usuario se pueden representar con clases UML y diagramas de colaboración. Si estamos ante invocaciones de servicios Web, se puede usar SoaML.

IFML se encarga de modelar:

- **La composición de la vistas**
- **El contenido de la vistas**
- **Las órdenes que puede dar el usuario**
- **Las acciones como respuesta a las órdenes del usuario**
- **Los efectos de la interacción del usuario y las consecuencias de sus acciones sobre el estado del interfaz**
- **Los parámetros entre los elementos del interfaz y las acciones a ejecutar**

- El modelo de objetos subyacente se puede representar con el clásico diagrama de clases de diseño UML.

El modelado con IFML aborda las siguientes perspectivas:

- **Especificación de la estructura de la vista.** Definición de contenedores, relaciones entre sí, su visibilidad, cómo llegar a ellos.
- **Especificación del contenido de la vista.** Definición de los componentes de la vista (contenido y vías de entrada de información)
- **Especificación de eventos.** Definición de los eventos que pueden afectar al estado del interfaz. Estos eventos pueden ser generados por la interacción del usuario, por la aplicación o por un sistema externo.
- **Especificación de la transición entre eventos.** La definición del efecto de un evento sobre el interfaz (cambio del contenedor o del contenido, desencadenamiento de una acción, o ambas cosas)
- **Especificación de los parámetros de conexión.** Definición de las dependencias de entrada/salida entre los componentes de la vista y entre éstos y las acciones.

Un diagrama IFML consiste en uno o más contenedores de vista de alto nivel. Por ejemplo, una aplicación de escritorio o una Rich Internet Application (RIA) se pueden modelar con un único contenedor de vista, la ventana principal. Por el contrario, una aplicación Web puede tener múltiples contenedores de alto nivel, uno para cada plantilla dinámica.

Cada contenedor de vista puede estructurarse internamente en una jerarquía de sub-contenedores. Por ejemplo, en una aplicación de escritorio la ventana principal puede dividirse en múltiples paneles o frames anidados u organizados en la pantalla, de forma que los contenedores hijos pueden visualizarse simultáneamente o de forma excluyente, en cuyo caso puede definirse uno de ellos como contenedor por defecto.

Un contenedor de vista puede contener componentes de vista, es decir, contenido o elementos de interfaz para la entrada de datos (por ejemplo, un formulario). Estos componentes de vista tienen parámetros de entrada y de salida:

- Un componente de vista que muestre las propiedades de un objeto tendría como parámetro de entrada el ID del objeto.
- Un formulario de entrada de datos, o una lista de ítems, tendría como parámetros de salida los valores introducidos o el ítem seleccionado.

Se pueden asociar componentes y contenedores de vista con eventos, para indicar que permiten la interacción con el usuario. Un componente de vista puede representar una lista con un evento asociado para seleccionar uno o más ítems, un formulario asociado a un evento de envío de la información o una galería de imágenes con un evento de scroll. Estos eventos se corresponden con acciones que dependen de la plataforma específica, y por tanto no se muestran en los diagramas IFML. Pensemos en el evento “pasar página” en una galería de imágenes: puede ser un enlace en

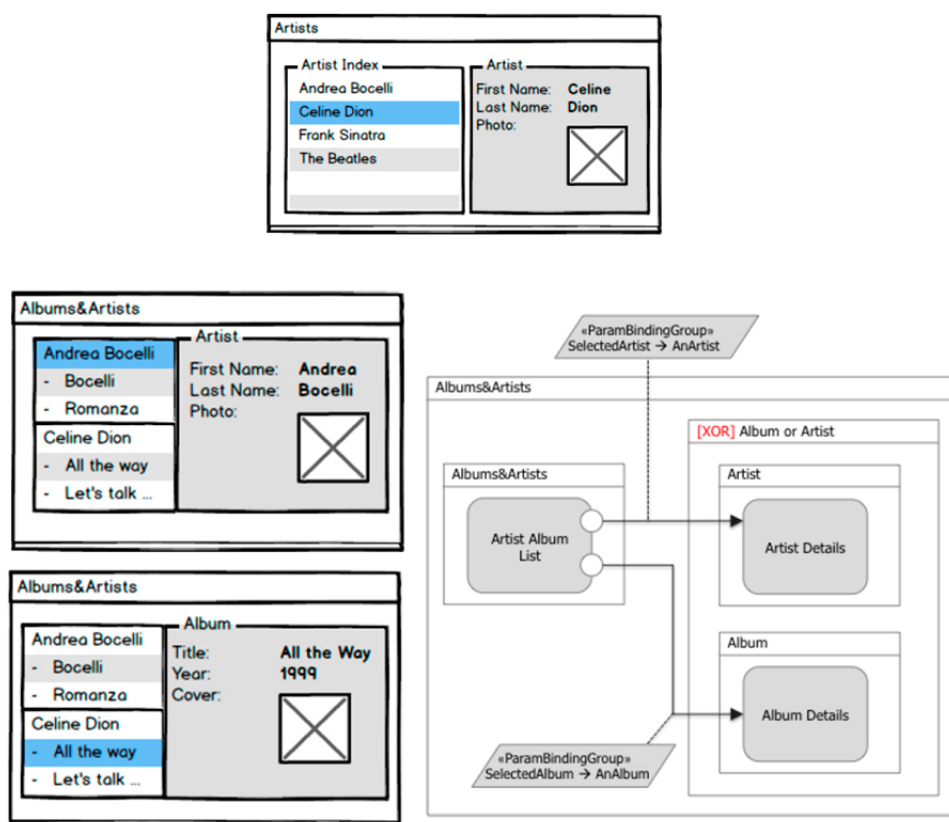


Ilustración 10: Interfaz de usuario con paneles exclusivos.

una página HTML o un gesto de deslizar dos dedos en una aplicación móvil.

Los efectos de los eventos se representan mediante una conexión entre el evento y el contenedor o componente de vista que se ve afectado, denominada flujo de interacción. Por ejemplo, en una aplicación Web es normal que al seleccionar un ítem de una lista se muestren los detalles del objeto seleccionado. Esto se representa, en el ejemplo de la ilustración 10, mediante un flujo de interacción que conecta el evento asociado al componente lista en un contenedor de alto nivel (la página Web) con el componente de vista que representa el detalle del objeto. El flujo de interacción expresa un cambio de estado en el interfaz de usuario.

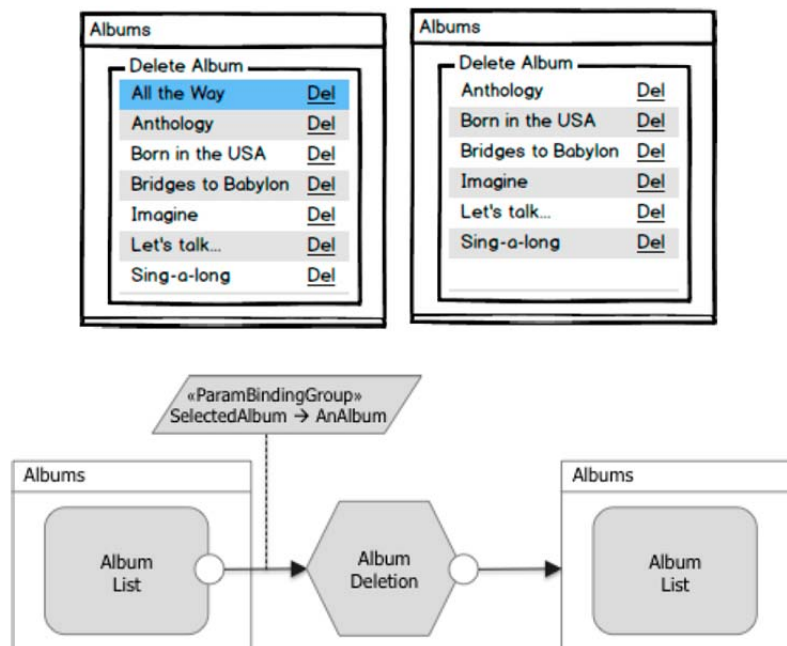

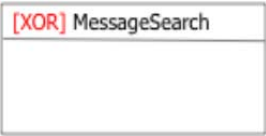
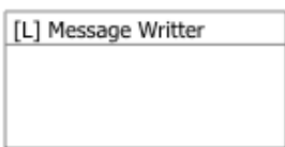
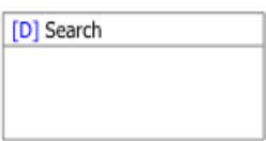
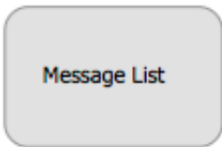





Ilustración 11: Ejemplo de interfaz de usuario con disparo de acción.



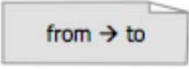

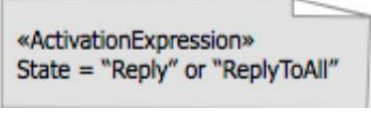
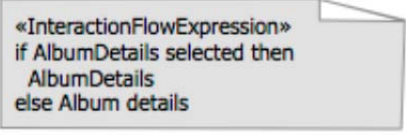



Un evento también puede *disparar una acción*, que se ejecuta antes de actualizar el estado del interfaz de usuario. Por ejemplo: el usuario selecciona en una lista los elementos a borrar, el evento de selección dispara la acción de borrar, tras la cual la página se refresca. El efecto de disparar una acción se representa con un flujo de interacción que conecta la acción con el contenedor o el componente de la vista afectado. Como ejemplo, en la figura 11 se muestra el diagrama IFML de una acción de borrado.

Una *dependencia E/S* entre elementos de vista, o entre elementos de vista y acciones, se denota mediante una *vinculación de parámetros* asociados a flujos de navegación. En la ilustración 10, vemos cómo el flujo de navegación que va desde el evento que denota la selección de un elemento en la lista de artistas hasta el componente de vista de un artista, tiene una *vinculación de parámetros* que asocia la salida del componente de vista “Lista de Artistas” con el parámetro de entrada del componente de vista “Artista”.

## Elementos IFML

| Concepto                                 | Significado  | IFML   | Ejemplo a nivel de implementación                          |
|--|--|--|--|
| <b>Contenedor de Vista</b>               | Un elemento del interfaz que contiene elementos que visualizan contenido o permiten la interacción o/y otros contenedores de vista                   |    | <i>Página Web<br/>Ventana<br/>Frame</i>                    |
| <b>Contenedor de Vista Excluyente</b>    | Un contenedor de vista que sólo muestra a la vez uno de los contenedores incluidos en él   |    | <i>Paneles Tabulados<br/>Div en HTML</i>                   |
| <b>Contenedor de Vista de Referencia</b> | Un contenedor de vista alcanzable desde cualquier otro elemento del interfaz de usuario en cualquier momento.  |    | <i>Un enlace "Logout" visible en todas las páginas.</i>    |
| <b>Contenedor de Vista por defecto</b>   | El contenedor de vista que será mostrado al usuario cuando éste visualice su contenedor padre.   |  | <i>Una página de bienvenida.</i>                           |
| <b>Componente de Vista</b>               | Un elemento del interfaz que muestra contenido o acepta entrada de datos.  |   | <i>Una lista HTML<br/>Un formulario<br/>Una galería JS</i> |
| <b>Evento</b>                            | Algo que afecta al estado de la aplicación   |   |  |
| <b>Acción</b>                            | Una funcionalidad de la lógica de negocio disparada por un evento; puede estar en el lado servidor (por defecto) o en el cliente (se pone [Cliente]) |   | <i>La actualización de BD<br/>El envío de un e-mail</i>    |
| <b>Flujo de navegación</b>               | Dependencia E/S. El origen de la flecha tiene alguna salida que se asocia con la entrada del destino.  |   | <i>Enviar y recibir parámetros en una petición HTTP</i>    |



|  |  |  |  |
|--|--|--|--|
| <b>Flujo de datos</b>                    | Transferencia de datos entre componentes de vista o acciones como consecuencia de una interacción,                 |     | <i>Drag &amp; Drop de información.</i>             |
| <b>Parámetro</b>                         | Un valor con tipo y nombre   |    | <i>ParámetrosGET/POST<br/>Variables Javascript</i> |
| <b>Asociación de parámetros</b>          | Especificación de que un parámetro de salida se asocia con una entrada de otro elemento.                           |     |  |
| <b>Grupo de asociación de parámetros</b> | Conjunto de parámetros de un flujo de datos o de navegación.   |    |  |
| <b>Expresión de activación</b>           | Expresión booleana asociada con un elemento de vista, un componente o un evento: si TRUE, el elemento se habilita. |    |  |
| <b>Expresión de flujo de interacción</b> | Determina cuál de los flujos de interacción se activan como consecuencia de un evento                              |  |  |
| <b>Módulo</b>                            | Parte del interfaz de usuario, con sus acciones, que puede ser reutilizado en otros entornos.                      |   |  |
| <b>Puerto de entrada</b>                 | Punto de interacción entre el módulo y su entorno, que recoge los parámetros que llegan al módulo.                 |   |  |
| <b>Puerto de salida</b>                  | Punto de interacción entre el módulo y su entorno, que recoge los parámetros que salen del módulo.                 |   |  |

|  |  |  |  |
|--|--|--|--|
| <b>Parte de un Componente de Vista</b> | Una parte de un componente de vida que no puede existir por sí mismo. Puede disparar eventos y tener flujos de entrada y salida. |  | <i>Campos de un formulario</i>                     |
| <b>Evento de selección</b>             | Evento que denota la selección de un ítem del interfaz   |  | <i>Selección de una fila de una tabla</i>          |
| <b>Evento de envío</b>                 | Evento que dispara un paso de parámetros   |  | <i>El envío de un formulario en HTML</i>           |
| <b>Lista</b>                           | Componente de vista usado para mostrar una lista de instancias de datos.   |  | <i>Tabla con filas de elementos del mismo tipo</i> |
| <b>Formulario</b>                      | Componente de vista para mostrar un formulario compuesto de campos   |  |  |
| <b>Detalles</b>                        | Componente de vista usado para mostrar detalles de una instancia específica de datos.  |  |  |
| <b>Ventana Bloqueante</b>              | Contenedor de vista mostrado en una nueva ventana que bloquea cualquier interacción con otros contenedores activos.              |  | <i>Un pop-up HTML</i>                              |
| <b>Ventana no Bloqueante</b>           | Contenedor de vista mostrado en una nueva ventana que bloquea cualquier interacción con otros contenedores activos.              |  |  |

## DISEÑO ARQUITECTÓNICO

El diseño arquitectónico intenta capturar las decisiones significativas sobre la organización de un sistema software, destacando los subsistemas que lo constituyen, sus componentes y la interacción entre ellos.

Un enfoque tradicional es definir una descomposición del software, representar la distribución de los componentes a diferentes niveles (p.ej. páginas estáticas y dinámicas, formularios, applets, scripts, etc.) y la relación entre ellos.

Para ello vamos a mostrar la solución propuesta en la Web Application Extension para UML (WAE). Según este método, el diseño de la arquitectura comienza con la definición de un diagrama de componentes, representando los componentes de la aplicación, sus interfaces, sus relaciones y las clases que representan el dominio del problema. Además de identificar las clases, el diseño consiste en clasificarlas en diferentes capas o elementos de la aplicación.

Las páginas de una Web pueden ser consideradas como objetos, y por tanto modeladas mediante clases UML. Pero claro, hay características que son propias de una aplicación Web, como por ejemplo: ¿tiene la página un script de servidor y por tanto su HTML se genera dinámicamente? La cosa se complica más si además hay funciones que sólo se ejecutan en el cliente.

Para suplir esta carencia, WAE utiliza dos estereotipos: <<static page>> y <<dynamic page>>. Las primeras son formularios (estereotipo <<Web form>>) o cualquier otro elemento incrustable, mientras que la lógica de negocio de la aplicación se modela en objetos del estereotipo <<serverpage>>. De esta forma, se delimita claramente lo que está en el cliente y lo que radica en el servidor.

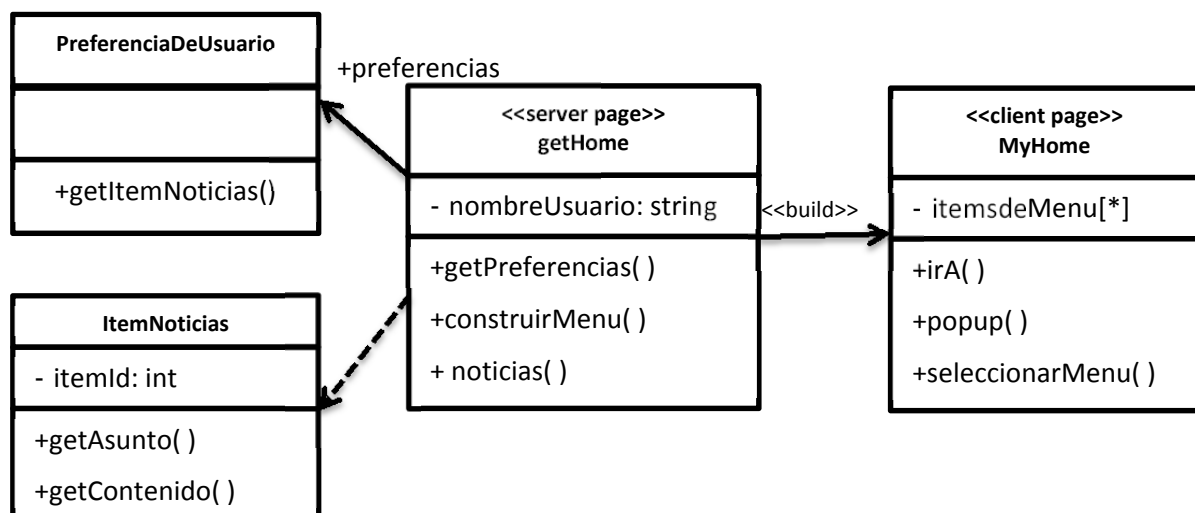


Ilustración 12: Diseño arquitectónico de una página Web personalizada.

Vamos a recordar, aunque es objeto de estudio en otra asignatura, algunos de los patrones arquitectónicos generales más comunes:

### Tuberías y filtros (pipe-and-filter)

Esta arquitectura se utiliza cuando el objetivo es tener un resultado tras varias etapas de procesamiento de una entrada determinada, y se ilustra gráficamente en la Ilustración 13. Los componentes computacionales funcionan como *filtros* que transforman la entrada en otros datos en función de unos algoritmos determinados. Los mecanismos de entrada y salida funcionan como *filtros* (pipes).



Ilustración 13: Arquitectura tubería-filtro

Los filtros han de ser componentes independientes. Es una de las bondades de esta arquitectura, que se pueden combinar filtros distintos, al estilo de la Shell de UNIX cuando encadenamos órdenes con el símbolo |.

Ejemplo: “Encontrar todos los anagramas del diccionario, es decir, agrupar las palabras que son permutaciones de las mismas letras (p.ej. ser y res).”

Solución siguiendo el patrón tubería y filtro: realizar tres componentes:

- Uno que ordene las letras de cada palabra de una lista, devolviendo otra lista.
- Otro que ordene las palabras de una lista, devolviendo otra lista.
- Otro que devuelva una lista con las palabras que estén repetidas en el listado ordenado que acepta como entrada..

### Modelo-Vista-Controlador

Este patrón es una forma de estructurar la aplicación en tres partes:

- El **modelo**, que gestiona uno o más elementos de datos, responde a cuestiones sobre su estado y realiza cambios de estado.
- La **vista**, que gestiona una zona del interfaz y es responsable de presentar datos al usuario y de recibir los eventos de teclado y ratón..
- El **controlador**, que gestiona todo el flujo de información en la aplicación, decidiendo quién hace cada cosa en cada momento.

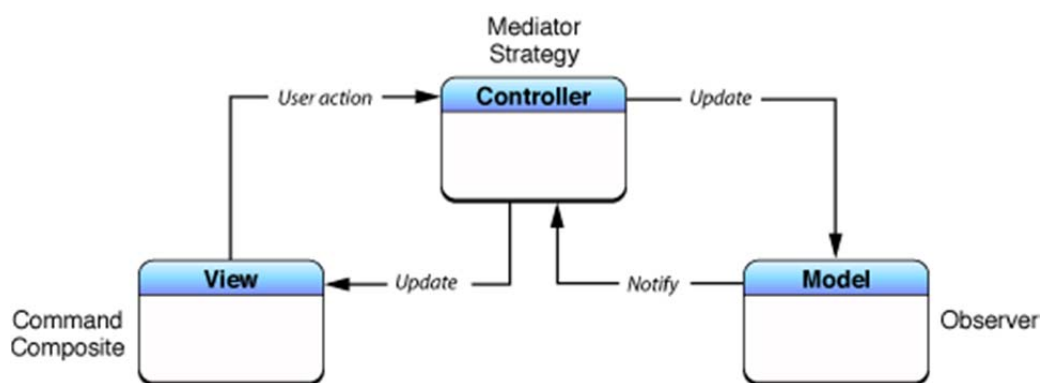


Ilustración 14 Modelo-Vista-Controlador

Cuando se usa un framework de desarrollo Web, p.ej. Symfony, Django o Yii2, todos obligan a seguir este patrón de desarrollo, por lo que debe ser una estrategia que usted ha de conocer y cultivar mucho a lo largo de sus estudios. Un esquema del MVC de Symfony se puede ver en la Ilustración 15.

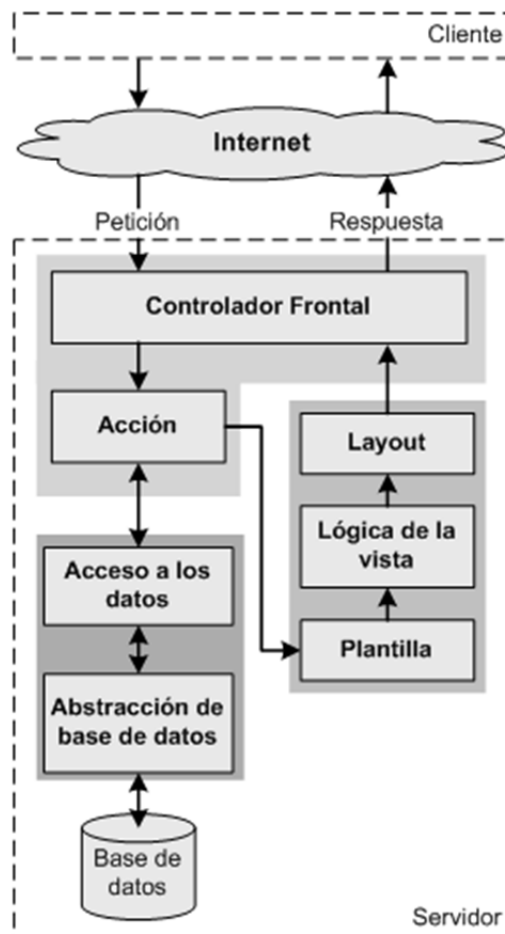


Ilustración 15: Flujo de trabajo de Symfony.

## DISEÑO DE LA ADAPTACIÓN

El desarrollo de aplicaciones que se adapten a distintas situaciones o necesidades es particularmente importante en la Web, ya que las aplicaciones están accesibles desde diversos lugares del mundo y se puede usar a través de muchos dispositivos diferentes.

Hay tres características en las que centrarse a la hora de diseñar la adaptabilidad de una aplicación basada en Web:

- *Localización e internacionalización.* La Web da la posibilidad de alcanzar una audiencia potencial casi gratis. Sin embargo, los contextos sociales, culturales e históricos de las comunidades de diferentes partes del mundo son muy distintos.
- *Personalización y adaptación.* Mientras que la localización se centra en cómo alcanzar un más amplio espectro de usuarios, basándonos en su singularidad cultural o demográfica, la personalización se centra en ajustar la aplicación Web a las necesidades del usuario particular, teniendo en cuenta su identidad.
- *Accesibilidad y usuarios con discapacidad.* Todo el mundo aprecia la tendencia actual que lleva hacia aplicaciones llenas de funcionalidad, pero hay gente con ciertas discapacidades funcionales como los invidentes, que encuentran dificultades enormes cuando tienen que navegar por simples páginas Web.

Las aplicaciones web que se pueden localizar, personalizar y que son accesibles normalmente se basan en un principio de ingeniería común: el principio de separación de responsabilidades. Si se desarrolla el software distinguiendo correctamente lo que es común a todos los usuarios, y lo que necesita de localización, personalización o adaptación, obtendremos un producto de mucha mayor calidad.



### Localización e internacionalización

El término *locale* se refiere a una región geográfica en la cual las personas comparten de cierta manera un idioma y unos valores (históricos, sociales, culturales) comunes.

Por otro lado, *internacionalización* ("i18n") o *inculturación* es el proceso de identificar y separar del resto del producto software todos los elementos específicos de cada uno de los *locale*, de forma que sea fácil adaptar para uno en concreto. Este proceso de adaptación es lo que se denomina *localización* ("l10n").

Por tanto, hay una estrecha relación entre la internacionalización y la localización: el propósito de la primera es hacer la localización más fácil, rápida, barata y de mayor calidad. La internacionalización no es fundamental para la localización, pero si se omite, y después queremos localizar nuestro desarrollo, será mucho más costoso.

Si estamos desarrollando un proyecto que sea global, posiblemente tengamos que localizar alguno de los siguientes elementos:

- *Idioma*. Es la diferencia más obvia entre los locales. Estamos ya muy acostumbrados a tener sitios multiidioma, e incluso que teniendo el mismo idioma cambie la terminología entre ellos (inglés EEUU o inglés GB).
- *Unidades de medida*. Mientras que en Europa se usa el sistema métrico decimal, en EEUU siguen anclados en las medidas victorianas (pulgadas, pies, etc.)
- *Moneda*. Las unidades monetarias suelen variar en cada país.
- *Formatos*. Muchas unidades se representan en diversos formatos según el *locale*: fechas (21-02-2013 vs 02-21-2013), números (3.14 vs 3,14), direcciones, etc.
- *Metáforas*. Algunas metáforas software son específicas de una zona, por ejemplo el concepto de *buzón de salida* o *mailbox*. En EEUU es muy normal dejar las cartas que se quieren enviar en el buzón y el mismo cartero las recoge para entregarlas ¿lo sabía?
- *Colores*. El significado de colores depende de la cultura. El rojo en la cultura china es buena suerte o celebración, mientras que en la cultura occidental suele asociarse con peligro.
- *Referencias históricas*. Referirse al Rey Sol en Europa es muy normal (todos sabemos que fue el rey francés Luis XIV) y por tanto comprender sus connotaciones, pero posiblemente otras culturas como la asiática o la árabe ignoren a qué no estamos refiriendo.
- *Aspectos culturales*. Un pulgar hacia arriba es OK o "bien" en algunas culturas, y un insulto en otras...

IFML y su herramienta CASE asociada (WebRatio) proporcionan soporte para el diseño conceptual y la implementación de aplicaciones web multilingües. Esto supone una serie de tareas como:



- **Diseño del modelo de datos** que permita la gestión de contenido multiidioma en la base de datos.
- **Especificación de modelos de selección** para que el usuario pueda escoger su *locale*.
- **Traducción de recursos estáticos** de la aplicación
- **La localización de las hojas de estilo** para adecuarlas a cada *locale*.



Ilustración 16: Personalización (tomado de [http://canterris.com/content\\_optimization](http://canterris.com/content_optimization))

## Personalización y adaptación

**Personalizar** una Web consiste en adaptar el sistema a un individuo, basándonos en su identidad y sus necesidades particulares. Ejemplos de tareas de personalización son mostrar las acciones más utilizadas por ese usuario concreto en un lugar más accesible, utilizar sus colores favoritos, etc. En realidad, la personalización es en cierto modo una **adaptación**.

La adaptación de una aplicación a determinados requisitos es una actividad que depende mucho de la correcta captura de los requisitos. En el caso de la personalización, en cierto modo es el usuario quien especifica sus preferencias, pero la adaptación en sí puede depender de otros factores como las características del dispositivo de visualización, el comportamiento en tiempo real del usuario (interpretando sus gustos indirectamente, como hace Amazon), etc.

En las aplicaciones Web de hoy día podemos encontrar los siguientes tipos de adaptaciones:

- *Adaptación de contenidos*: el contenido (los datos publicados en la página) se adapta para aumentar la efectividad del mismo, bien escogiendo diferentes presentaciones del mismo elemento (abreviado o largo) o seleccionando uno u otro ítem, diferentes niveles de detalle, etc.
- *Acciones de navegación automática*: en lugar de adaptar el contenido, se redirige al usuario de forma automática a otra página que le puede servir mejor a sus necesidades.
- *Adaptación de la estructura de navegación*: los enlaces se pueden mostrar u ocultar en función de los privilegios del usuario, o de las características de la operación que se esté realizando.
- *Adaptación del layout*: el layout (la organización de contenido y elementos gráficos de la página) se adaptan, normalmente, para encajar mejor a las resoluciones de pantalla y tamaños de ventana. Se suele conseguir utilizando hojas de estilo distintas o cambiando dinámicamente las propiedades de los elementos de la página.