UNIVERSIDAD NACIONAL DE SANTIAGO DEL ESTERO
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍAS

LICENCIATURA EN SISTEMAS DE INFORMACIÓN

TRABAJO FINAL DE GRADUACIÓN

ARQUITECTURA SOFTWARE REUTILIZABLE BASADA EN PATRONES DE DISEÑO Y PATRONES DE INTERACCIÓN, PARA EL DESARROLLO RÁPIDO DE APLICACIONES WEB

Autor
MARTÍN MIGUEL MURILLO

Profesora Guía
MSC. ING. ROSA PALAVECINO DE GÓMEZ

Septiembre 2009

UNIVERSIDAD NACIONAL DE SANTIAGO DEL ESTERO
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍAS

TRABAJO FINAL DE GRADUACIÓN DE LA LICENCIATURA EN SISTEMAS DE INFORMACIÓN

ARQUITECTURA SOFTWARE REUTILIZABLE BASADA EN PATRONES DE DISEÑO Y PATRONES DE INTERACCIÓN, PARA EL DESARROLLO RÁPIDO DE APLICACIONES WEB

Autor

.....
Martín Miguel Murillo

Profesor Guía

.....
MSc. Ing. Rosa Palavecino de Gómez

Aprobado el día ..... **del mes de** ..... **del año 20**.....

por el Tribunal integrado por

.....	**.....**	**.....**
.....	**.....**	**.....**

Santiago del Estero - Argentina

A mis padres Mario y Luisa

A Ivana María Luján

Martín Miguel Murillo

Septiembre 2009

Agradecimientos

Gracias a mis padres -*mis mayores ejemplos*- por haberme dado todo y mucho más aún. Gracias por los valores inculcados, por su paciencia infinita y su apoyo incondicional en cada una de mis decisiones.

Gracias a Rita Beatriz, por haberme enseñado en más de una oportunidad el camino durante mi vida de estudiante.

Gracias a Ivana María Luján, por estar a mi lado, preocuparse siempre, motivarme y compartir sueños de un futuro juntos.

Gracias a Rosy Palavecino, por haber creído en este proyecto, por su amistad, confianza y sus sabios consejos.

Por último, agradezco a todas aquellas personas -*familiares, amigos, profesores, excompañeros de estudio y actuales compañeros de trabajo*- que dieron su tiempo, comprensión, palabras de aliento, y que de alguna u otra manera ayudaron con el logro de este trabajo.

Martín Miguel Murillo

Septiembre 2009

ÍNDICE DE CONTENIDOS

Resumen

Lo que se busca obtener en el presente trabajo, es una herramienta que permita minimizar tiempo y costos en el desarrollo de aplicaciones Web, a través de un diseño reutilizable con soporte a las últimas tecnologías y estándares abiertos. Para ello, se propone el desarrollo de un framework basado en patrones de diseño y patrones de interfaces, como solución a los problemas comunes de desarrollo software, y a los problemas repetidos de diseño de interfaces de usuario, a partir del cual sea posible generar o extender Aplicaciones Web funcionales en forma rápida y sencilla, mediante un enfoque práctico, intentando constituir así un aporte al diseño de arquitecturas software reutilizables.

El framework obtenido, fue utilizado en la generación de un conjunto de aplicaciones Web para la gestión de contenidos periodísticos de nuestro medio.
Mediante el uso de COCOMO II, se estimó a priori, la duración y el número de programadores/mes necesarios para uno de los proyectos. Se utilizó además COCOMO tradicional, para calcular el tiempo y el esfuerzo de un desarrollo similar en cuanto a la cantidad de líneas de código de la aplicación extendida del framework. La idea, fue la de contrastar el tiempo y esfuerzo de lo que podría considerarse un desarrollo similar con métodos tradicionales, contra los valores reales obtenidos. En ambos casos, los valores reales fueron más bajos, siendo los arrojados por COCOMO II, los valores más cercanos.

Se evaluaron además, la facilidad de uso y el grado de aceptación del framework por parte de un conjunto de programadores, y dos atributos de calidad, como la usabilidad y facilidad de mantenimiento de las aplicaciones extendidas del framework, obteniendo en todos los casos resultados satisfactorios.

Palabras Clave:

Arquitectura Software, Framework, Aplicaciones Web, Patrones de diseño, Patrones de interface.

Introducción

La Revolución Industrial, introdujo profundos cambios en los procesos productivos de las grandes industrias. Productos que hasta hace un tiempo se venían fabricando de manera artesanal, comenzaron a elaborarse a través de procesos automatizados que permitieron aumentar considerablemente la productividad minimizando recursos.

En el ámbito de la Ingeniería del Software, y haciendo una analogía con la llamada *Revolución Industrial*, las arquitecturas reusables y los frameworks, han marcado de alguna manera una revolución y una alternativa de cambio en el proceso de desarrollo del software. Tal vez no falte demasiado, para que resulte común disponer de sistemas generadores de aplicaciones a medida, verdaderas cajas negras que permitan obtener en unos pocos clicks y en cuestión de minutos, aplicaciones totalmente funcionales orientadas a un dominio y necesidades específicas. De hecho, ya existen en el mercado potentes herramientas basadas en estos principios, pero su principal dificultad radica en el excesivo costo de las licencias, y en la necesidad de contar aún con expertos que asistan en el uso de las mismas.

Es en este sentido, y salvando las distancias, lo que se propone en el presente trabajo, es desarrollar, probar y posteriormente evaluar una arquitectura software reutilizable. Un framework basado en patrones de diseño, y patrones de interface, que permita generar de forma rápida y sencilla, aplicaciones Web para la gestión de contenidos.

Capítulo 1

Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web

Problema y Objetivos

I.1. Introducción

El presente trabajo plantea el diseño y desarrollo de un *Framework*[1] basado en *patrones de diseño*[2] y *patrones de interacción*[3], para la generación rápida de aplicaciones Web, facilitando una estructura y una metodología de trabajo que permitan extender, organizar, y desarrollar otro proyecto software en forma rápida y sencilla, dejando para el desarrollador sólo cuestiones que hacen a los requerimientos funcionales de la aplicación extendida del modelo. En éste sentido:

- Se analizan las ventajas e inconvenientes del uso de éste tipo de herramientas,
- Se estudian los distintos Patrones de Diseño conocidos, como solución a los problemas que se plantean en el desarrollo de este tipo de modelos,
- Se propone el uso de PHP5 como lenguaje Orientado a Objetos para la implementación de las diferentes clases y patrones de diseño.
- Se propone un conjunto de patrones de Interface de Usuario o patrones de interacción, basados en el estándar XML, como solución a problemas comunes de interacción.

A continuación se delimita y se plantea el problema, se expone una síntesis de la exploración bibliográfica realizada, y se expresa la finalidad y los objetivos trazados al comienzo. Por último, se procede a especificar el alcance del trabajo como así también su grado de desarrollo.

---

[1] Un Framework es un conjunto de clases cooperantes que constituyen un diseño reutilizable para una clase específica de software. Un Framework representa las decisiones de diseño que son comunes a su dominio de aplicación, y hacen hincapié en la reutilización del diseño, frente a la reutilización del código.

[2] Los Patrones de diseño fueron originalmente propuestos por Christopher Alexander en el contexto del diseño y construcción urbanística. Partiendo de la definición original de Alexander, un patrón de diseño es una solución a un problema que se usa repetidamente en contextos similares con algunas variantes en la implementación.

[3] Al igual que los patrones de diseño aportan soluciones a problemas comunes de desarrollo software, los patrones de interfaces de usuario (o patrones de interacción) intentan dar soluciones efectivas a problemas repetidos de diseño de interfaz, constituyendo una alternativa válida para el desarrollo de interfaces usables en ambientes WEB

I.2. Planteamiento del Problema

Los patrones de diseño, constituyen una disciplina reciente de la Ingeniería del Software, resultante de las experiencias continuas de enfrentarse a determinados problemas. Su objetivo es dar soluciones estandarizadas y probadas a problemas comunes de desarrollo, ofreciendo mejores prácticas en torno al problema a resolver favoreciendo la escalabilidad de un producto software [5].

Al igual que en el campo de la Ingeniería del Software, los patrones de diseño también se aplican en el área del Diseño de Interfaces de Usuario e Interacción Persona-Ordenador, pudiendo estos dos modelos complementarse y ser usados conjuntamente en el diseño de aplicaciones interactivas [14, 16, 10, 21].

Los Frameworks, son modelos o estructuras de software que permiten organizar y desarrollar otro proyecto software, modelando las relaciones generales de las entidades de un dominio, facilitando la estructura y metodología de trabajo que las aplicaciones de dicho dominio extienden o utilizan. Los frameworks pueden implementar uno o más patrones de diseño software, y disponer también de un conjunto de patrones de interface o patrones de interacción. En el ámbito de las aplicaciones Web, por lo general es implementado el patrón MVC (Modelo-Vista-Controlador) que separa en capas [22]:

1) la lógica de negocios (El Modelo),
2) la presentación final al cliente de los datos procesados (La Vista), y
3) la capa encargada de recibir las entradas de usuario, para convocar a los métodos adecuados del modelo y de las vistas (El Controlador)

Actualmente existen frameworks basados en diferentes lenguajes y plataformas. Algunos más complejos que otros, presentan ventajas y desventajas entre sí al momento de evaluar su aplicación y uso, sobre todo cuando se requieren resultados a medida, y en períodos relativamente cortos de tiempo:

1) Los frameworks del tipo Open Source, generalmente no cuentan con documentación suficiente y detallada para encarar a corto plazo un proyecto relativamente complejo.
2) Muchos no implementan soluciones basadas en tecnologías de capas, como el patrón MVC [22].

3) No todos los frameworks existentes soportan tecnologías como AJAX [3].
4) La mayoría de los frameworks son demasiados generales ó demasiados específicos para dar un soporte adecuado y a medida, de acuerdo a las características especificas de una determinada aplicación.
5) Los frameworks comerciales requieren de licencias relativamente costosas en cuanto al software de soporte para las aplicaciones generadas

En función de los puntos mencionados, todo el esfuerzo necesario para adoptar y/o adaptar alguno de los modelos ya existentes, incluidos evaluación, costos de licencias, de soporte, recursos humanos, plazos de desarrollo entre otros, plantean la posibilidad de diseñar un nuevo modelo ajustado a la medida de las necesidades.

I.3. Antecedentes

El término framework, se refiere a una estructura software formada por componentes personalizables para el desarrollo de una aplicación determinada. Se lo puede considerar como una aplicación genérica y configurable, a la que es posible añadir las últimas piezas para construir una aplicación concreta [17].

Íntimamente relacionado, se encuentran los patrones de diseño, como una importante técnica para construir software orientado a objetos [5]. Los patrones ofrecen una solución para un problema de diseño común, describiendo cómo una sociedad de clases (y objetos) trabajan juntos para resolver ese problema en un contexto particular [14].

Un framework que se considere maduro, implementa uno ó más patrones de diseño, ya que estos pueden considerarse elementos arquitectónicos de los frameworks, lo cual los hace más abstractos [1].

Actualmente existen una infinidad de frameworks orientados a distintos dominios [5]: como la construcción de editores gráficos, construcción de compiladores, aplicaciones de modelado financiero, aplicaciones de gestión de contenidos, etc.

[1] Los frameworks pueden plasmarse en código mientras que solo los ejemplos de patrones pueden implementarse mediante código. Además, los Frameworks están orientados a un dominio de aplicación específico mientras que los patrones son más generales.

En el ámbito de las aplicaciones Web, se destacan, los frameworks orientados a la interfaz de usuario, a las aplicaciones de publicación de documentos, al control de eventos, comercio electrónico, entre otros.

A continuación se destacan 3 de los frameworks más utilizados en el desarrollo de aplicaciones Web bajo el patrón MVC.

- **Struts** es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC sobre plataformas J2EE (Java 2, Enterprise Edición). Actualmente es un proyecto independiente conocido como **Apache Struts**
- **.NET Framework** es el componente de Windows para crear y ejecutar aplicaciones de software y servicios Web XML. Se encarga de la mayor parte de la estructura necesaria para generar software, permitiendo a los programadores centrarse en el código lógico esencial para el negocio.
- **Zend Framework** es una herramienta para el desarrollo de aplicaciones Web basadas en PHP bajo en patrón MVC.

Resulta importante hacer notar que los frameworks antes mencionadas constituyen potentes herramientas de desarrollo de propósito general, mientras que la arquitectura propuesta en el presente trabajo, presenta una solución estandarizada a problemas repetidos de diseño, dentro del dominio de las aplicaciones Web para la gestión de contenidos.

Otro aspecto a tener en cuenta, es el soporte AJAX [3] por parte de la mayoría de los frameworks y herramientas de desarrollo Web comerciales existentes, como una forma de reducir las limitaciones de interacción propias de un ambiente Web. Se trata de una técnica de desarrollo para la creación de aplicaciones interactivas mediante la combinación de tres tecnologías existentes:

1) HTML para presentar la información,
2) Document Object Model (DOM) y JavaScript, para interactuar dinámicamente con los datos, y
3) XML, para el intercambio de datos asincrónicamente con un servidor web.

Es tal vez Google una de las referencias más importantes al momento de mencionar proyectos que utilizan AJAX en el desarrollo de aplicaciones Web comerciales:

- **Google Groups:** Comunidad de grupos de usuarios de google. http://groups-beta.google.com
- **Google Suggest:** buscador google con búsqueda sugerida. http://www.google.com/webhp?complete=1&hl=en
- **Gmail**: Servicio de correo de google. http://www.gmail.com
- **Google Maps**: Sistema de Información Geográfico de Google, http://web.archive.org/web/20050331002145/jgwebber.blogspot.com/2005/02/mapping-google.html

Los proyectos antes mencionados, demuestran que AJAX no se trata sólo de una técnica más, sino que se trata de un conjunto de tecnologías bien probadas, aplicables a casi cualquier problema del mundo real.

I.4. Objetivos de la Investigación

Objetivo General

Desarrollar una arquitectura software reusable o Framework [23], basado en patrones de diseño, y patrones de Interface de usuario, para la generación rápida de Aplicaciones Web multiplataforma.

Objetivos Específicos

- Facilitar la tarea del programador en el desarrollo de aplicaciones Web.
- Minimizar los tiempos de desarrollo.
- Brindar un mayor grado de mantenibilidad, robustez, y usabilidad en el desarrollo de aplicaciones.
- Proponer patrones de Interface de Usuarios, basados en el estándar XML como solución a los problemas comunes de la interacción.

I.5. Alcance

El presente trabajo implica el diseño, desarrollo y validación de una arquitectura software reusable para la generación rápida de aplicaciones Web basadas en PHP [18, 19], que resuelvan cuestiones de diseño generales tales como:

a) Arquitecturas de capas basada en el patrón MVC [22]
b) Interfaces estandarizadas por un conjunto de patrones que determinan la forma en que los usuarios interactúan con el sistema extendido.
c) Soporte AJAX. [3]
d) Soporte MySQL

El framework obtenido, se utilizará para el desarrollo de un prototipo Web de Gestión de Contenidos *(Web Content Managment System, WCMS[1])* para la administración de los contenidos de un sitio del tipo NewsLetter o Noticias.

[1] Un sistema de administración de contenidos permite la creación y administración de los contenidos de sitios web, y básicamente consiste de en una interfaz que controla una o varias bases de datos donde se alojan los contenidos.

Capítulo 2

Arquitectura software reutilizable basada en patrones de diseño y
patrones de interacción, para el desarrollo rápido de aplicaciones web

Marcos Referenciales

II.1. Introducción

En este capítulo, se introducen las reseñas de las diferentes teorías y tecnologías en las que se basa el presente trabajo, las técnicas que permitirán evaluar y contrastar los resultados obtenidos, y por último, el universo de estudio y el producto final que se pretende obtener a partir de éste.

II.2. Marco Teórico

II.2.1 Patrones de diseño

Según Christopher Alexander, *"cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces"*. Aunque Alexander se refería a patrones en ciudades y edificios, lo que dice también es válido para patrones de diseño orientado a objetos. Y en este caso las soluciones se expresan en términos de objetos e interfaces en vez de paredes y puertas, pero en la esencia de ambos tipos de patrones se encuentra una solución a un problema dentro de un contexto [5].

En general, un patrón tiene cuatro elementos esenciales [5]:

1. El **nombre del patrón** permite describir, en una o dos palabras, un problema de diseño junto con sus soluciones y consecuencias. Al dar un nombre a un patrón se está inmediatamente incrementando el vocabulario de diseño, lo que permite diseñar con mayor abstracción.

2. El **Problema** describe cuando aplicar el patrón. Explica el problema y su contexto. Puede describir problemas concretos de diseño. A veces el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón.

3. La **solución** describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o una implementación en concreto, sino el hecho de que un patrón resulta una especia de plantilla aplicable en muchas situaciones diferentes.

4. Las **consecuencias** son los resultados así como las ventajas e inconvenientes de aplicar el patrón. Aunque cuando se describen decisiones de diseño muchas veces no se reflejan consecuencias, éstas resultan fundamentales para evaluar las alternativas de diseño y comprender los costos y beneficios de aplicar el patrón.

II.2.1.1 Descripción de los patrones de diseño [5]

Las notaciones gráficas, aunque importantes no son suficientes. Simplemente representan el producto final del proceso de diseño, como las relaciones entre clases y objetos. Para reutilizar el diseño, debemos hacer constar las decisiones, alternativas, ventajas e inconvenientes que dieron lugar a él. También resultan importantes los ejemplos concretos, ya que ayudan a ver el diseño en acción.

Se describen los patrones empleando un formato consistente. Cada patrón se divide en secciones la cual da una estructura uniforme a la información, haciendo que los patrones de diseño sean más fáciles de aprender, comparar y usar.

Nombre del patrón y clasificación. El nombre del patrón transmite su esencia. Un buen nombre es vital porque pasará a formar parte del vocabulario de diseño

Propósito. Una frase breve que responde a las siguientes cuestiones: ¿Qué hace este patrón de diseño? ¿En qué se basa? ¿Cuál es el problema de diseño concreto que resuelve?

También conocido como. Otros nombres, si existen, por los que se conoce al patrón.

Motivación. Un escenario que ilustra el problema y cómo las estructuras de clases y objetos del patrón resuelven el problema.

Aplicabilidad. ¿En qué situaciones se puede aplicar el patrón de diseño? ¿Qué ejemplos hay de malos diseños que el patrón puede resolver?

Estructura. Una representación gráfica de las clases del patrón usando una notación basada en la técnica de modelado de objetos (OMT), y

mediante el uso de diagramas de interacción para mostrar secuencia de peticiones y colaboraciones entre objetos.

Participantes. Las clases y objetos participantes en el patrón de diseño, junto con sus responsabilidades.

Colaboraciones. Cómo colaboran los participantes para llevar a cabo sus responsabilidades.

Consecuencias. ¿Cómo consigue el patrón sus objetivos? ¿Cuáles son las ventajas e inconvenientes y los resultados de usar el patrón? ¿Qué aspectos de la estructura del sistema se pueden modificar de forma independiente?

Implementación. ¿Cuáles son las dificultades, trucos y técnicas que se deberían tener en cuenta a la hora de aplicar el patrón? ¿Hay cuestiones específicas del lenguaje?

Código de ejemplo. Fragmentos de código que muestren cómo se puede implementar el patrón.

Usos conocidos. Ejemplos del patrón en sistemas reales. Al menos dos ejemplos de dominios diferentes.

Patrones relacionados. ¿Qué patrones de diseño están estrechamente relacionados con éste? ¿Cuáles son sus principales diferencias? ¿Con que otros patrones debería usarse?

II.2.1.2 Catálogo de patrones de diseño [5]

A continuación, se describe la colección de los 23 patrones expuesta por Erich Gamma en su libro, “Patrones de diseño”:

Abstract Factory (Fábrica Abstracta). Proporciona una interfaz para crear familias de objetos relacionados o que dependan entre sí, sin especificar sus clases concretas.

Adapter (Adaptador). Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permite que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

Bridge (Puente). Desacopla una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

Builder (Constructor). Separa la construcción de un objeto complejo de su representación, de forma que el proceso de construcción pueda crear diferentes representaciones.

Chain of responsability (Cadena de responsabilidad). Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que ésta sea tratada por alguno de los objetos.

Command (Orden). Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.

Composite (Compuesto). Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

Decorator (Decorador). Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

Facade (Fachada). Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

Factory Method (Método de Fabricación). Define una interfaz para crear un objeto, pero que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue a sus subclases la creación de objetos.

Flyweight (Peso ligero). Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

Interpreter (Intérprete). Dado un lenguaje, define una representación de su gramática junto con un intérprete para interpretar las sentencias del lenguaje.

Iterator (Iterador). Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

Mediator (Mediador). Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

Memento (Recuerdo). Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste pueda regresar a sus estado anterior más tarde.

Observer (Observador). Define una dependencia de uno a muchos entre objetos, de modo que cuando un objeto cambie de estado se notifica y actualizan todos los objetos que dependen de él.

Prototype (Prototipo). Especifica los tipos de objetos a crear por medio de una instancia protitípica, y crea nuevos objetos copiando de éste prototipo.

Proxy (Apoderado). Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Singleton (Único). Garantiza que una clase tenga sólo una instancia, y proporciona un punto de acceso global a ella.

State (Estado). Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Parecerá que cambia la clase del objeto.

Strategy (Estrategia). Define una familia de algoritmos, encapsula a cada uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.

Template Method (Método Plantilla). Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus

pasos. Permita que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.

Visitor (Visitante). Representa una operación sobre los elementos de una estructura de objetos. Permite definir una operación sin cambiar las clases de los elementos sobre los que opera.

II.2.1.3 Clasificación de patrones de diseño [5]

Patrones de diseño				
Propósito				
		De creación	**Estructurales**	**De comportamiento**
Ámbito	**Clase**	Factory Method	Adapter *(de clases)*	Interpreter Template Method
	Objeto	Abstract Method Builder Prototype Singleton	Adapter *(de objetos)* Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsability Command Iterator Mediator Memento Observer State Strategy Visitor

Como se aprecia en la tabla precedente, se clasifican los patrones siguiendo dos criterios. El primero de ellos denominado **propósito**, refleja que hace cada patrón. Los patrones pueden tener un propósito de **creación**, **estructural** o **de comportamiento**. Los patrones de creación tienen que ver con el proceso de creación de objetos. Los patrones estructurales tratan con la composición de clases u objetos. Los de comportamiento caracterizan el modo en que las clases y objetos interactúan y se reparten las responsabilidades.

El segundo criterio, denominado **ámbito**, especifica si el patrón se aplica principalmente a clases o a objetos. Los patrones de clases se ocupan de las relaciones entre clases y sus subclases. Estas relaciones se establecen a través de la herencia, de modo que son relaciones estáticas, fijadas en tiempo de compilación. Los patrones de objetos tratan con las relaciones entre objetos, que pueden cambiarse en tiempo de ejecución y son más dinámicas. Casi todos los patrones usan la herencia de un modo u otro, así que los únicos patrones etiquetados como “patrones de clases” son aquellos que se centran en las relaciones entre clases.

II.2.1.4 El patrón de arquitectura MVC (Modelo-Vista-Control) [22]

MVC fue introducido inicialmente en la comunidad de desarrolladores Smalltalk-80. MVC divide una aplicación interactiva en 3 áreas: de procesamiento, salida y entrada, utilizando para esto las siguientes abstracciones:

- **Modelo.** Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y comportamiento de entrada.
- **Vista.** Muestra la información al usuario. Obtiene los datos del modelo. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- **Controlador.** Recibe las entradas, usualmente como eventos. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista. El usuario interactúa con el sistema a través de los controladores.

Las **Vistas** y **Controladores** conforman la interfaz de usuario. Mientras que un mecanismo de propagación de cambios asegura la consistencia entre la interfaz y el modelo.

La separación del modelo de los componentes vista y del controlador permite mantener múltiples vistas del mismo modelo. Si el usuario cambia el modelo a través del controlador de una vista, todas las otras vistas dependientes deben reflejar los cambios. Por lo tanto, el modelo notifica a todas las vistas siempre que sus datos cambien. En la Figura II.1 se muestra el diagrama de clases del patrón MVC [22].

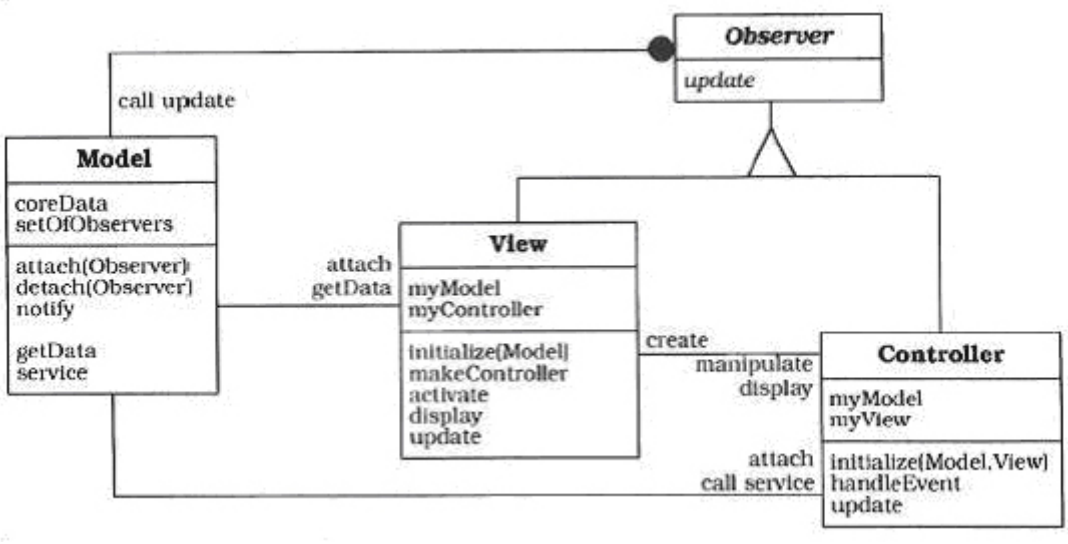


Figura II.1 - diagrama de clases del patrón MVC [22]

El patrón de arquitectura ha sido portado a una gran cantidad de entornos y frameworks como entre los que se encuentran WinForms, ASP.Net, etc. Herramientas de programación como Visual Basic, Visual Studio .Net, etc., siguen también alguna variante de este esquema.

El MVC es un patrón ampliamente utilizado en múltiples plataformas y lenguajes. Y algunos de sus principales beneficios son:

- **Menor acoplamiento**
  - Desacopla las vistas de los modelos
  - Desacopla los modelos de la forma en que se muestran e ingresan los datos
- **Mayor cohesión**
  - Cada elemento del patrón esta altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio)
- **Las vistas proveen mayor flexibilidad y agilidad**
  - Se puede crear múltiples vistas de un modelo
  - Se puede crear, añadir, modificar y eliminar nuevas vistas dinámicamente
  - Las vistas pueden anidarse
  - Se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual

    - Se puede sincronizar las vistas
    - Las vistas pueden concentrarse en diferentes aspectos del modelo
- **Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales**
    - Una vista para cada dispositivo que puede variar según sus capacidades
    - Una vista para la Web y otra para aplicaciones de escritorio
- **Más claridad de diseño**
- **Facilita el mantenimiento**
- **Mayor escalabilidad**

Un patrón de arquitectura puede implementar uno o más patrones de diseño. Para el caso del MVC en el cual se centra el presente trabajo, se implementarán los siguientes patrones de diseño:

- **Abstract Factory (Fábrica Abstracta).** Para crear familias de objetos (los diferentes patrones de interacción) sin especificar sus clases concretas.
- **Observer (Observador).** Para definir un conjunto de eventos que se disparan ante determinadas circunstancias, y cuyo código puede variar según las necesidades o reglas de negocio.

II.2.2 Patrones de Interacción [10, 21]

La aceptación final de una aplicación software por parte de un usuario depende en gran medida de la percepción que éste tenga del sistema y esta percepción se logra mediante la interface del sistema.

En la literatura correspondiente al diseño de interfaces de usuario se habla mucho de la importancia de diseñar interfaces usables, (fáciles de aprender, de usar, robustas, flexibles etc.) sin embargo, la forma de incorporar esta característica en los diseños resulta poco clara. Es por eso que se plantea el uso de Patrones de Interacción para el diseño de interfaces usables, en virtud de que los patrones especifican claramente la forma de implementación, el contexto en el que pueden ser aplicados e incluso las consecuencias de su uso, sin olvidar que los Patrones de Interacción están basados sobre principios de Usabilidad.

El primer intento por aplicar este concepto en el diseño de las interfaces de usuario se dio por Ward Cummingham y Kent Beck quienes crearon cinco patrones de interfaz: Window per task, Few panes, Standard panes, Nouns and verbs, y Short Menu [10].

Sin embargo no fue hasta finales de los noventa en que irrumpen en escena los Patrones de Interacción siendo estos más cercanos a las ideas del Arquitecto Christopher Alexander [10], quien propuso el paradigma de patrones teniendo en mente una cuestión de estética y confort en la construcción de edificios y de considerar al usuario del edificio como parte del equipo de diseño del mismo, en virtud de que es él quien conoce mejor que nadie los requerimientos.

II.2.2.1 Usabilidad

Es muy frecuente encontrar interfaces WEB en las que no se ha considerado que el usuario no es la mayoría de las veces un experto, sino que la aplicación puede ser usada por un publico diverso. Por lo tanto, incluir solo características estéticas en una interfaz y no considerar la usabilidad de la misma, provoca un fuerte desaliento en el usuario final.

Lo deseable es que las interfaces de aplicaciones sean fáciles de usar, de navegar, agradables al usuario, que contengan elementos bien distribuidos, y con la información que el usuario espera.

La usabilidad es “La medida en la que un producto se puede usar por determinados usuarios para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso particular”. Una interfaz de usuario usable es fácil de aprender para los nuevos usuarios, es flexible porque el usuario y el sistema pueden intercambiar información de varias formas, es sólido porque es posible utilizar el sistema sin problemas ya que puede recuperarse de los errores y responder en un tiempo razonable [10].

Si bien existen una gran cantidad guías, pautas, estándares y reglas ergonómicas para diseñar interfaces usables, generalmente no resulta fácil ponerlas en práctica ya que no se precisa el escenario bajo el cual pueden aplicarse, además de que no se conocen las consecuencias de su uso y no se tienen ejemplos específicos de cómo emplearlas [10].

Un patrón en cambio tiene un formato bien definido, y sus atributos frecuentes son: nombre del patrón, problema que resuelve, solución propuesta, contexto, ejemplos, etc. A continuación se presenta, en términos generales, la estructura de los patrones de la colección de Van Wellie [10]:

Nombre. El título del patrón, el cual debe ser representativo, claro y conciso del concepto a comunicar.

Autor. Quien propone al patrón.

Problema. Una descripción del problema desde el punto de vista del usuario.

Principio de usabilidad. Describe los principios o criterios de usabilidad en los cuales se basa el patrón.

Contexto. Una descripción de la situación en la cual puede usarse el patrón, cuales son las características del contexto, en términos de las tareas, del usuario.

Fuerza. Aspectos del contexto que necesitan ser optimizados.

Solución. Descripción clara de la solución propuesta (otros patrones pueden ser necesarios para completar la solución completa del problema)

Consecuencias. Describe los resultados de aplicar el patrón.

Ejemplo. Un ejemplo ilustrativo de una solución exitosa.

Para mayor claridad del concepto de patrón de interacción, a continuación se muestra en la Figura II.2 un caso particular:

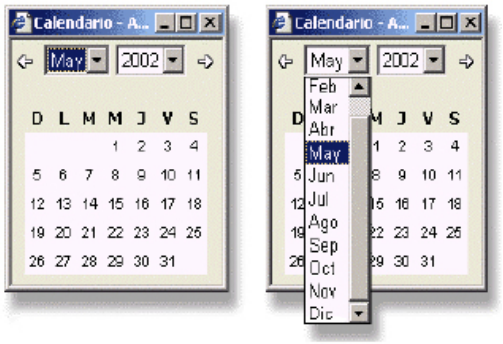


Figura II.2 – Formato de datos de fechas

Nombre. Formatos de Datos de fechas.

Autor. Martin Van Wellie

Problema. El usuario desea introducir datos de fechas y no desea preocuparse por la sintaxis del dato.

Principio de Usabilidad. Guiar al usuario y prevenir errores

Contexto. Todos los sistemas que requieran que el usuario introduzca fechas

Fuerzas. Los datos de fechas tienen múltiples sintaxis.

Solución. Permitir que el usuario elija la fecha de un calendario como en el mundo real.

Ejemplo. Realizar la búsqueda de algún archivo creado a partir de una fecha el calendario le ayudará a llenar las casillas de la fecha con el formato de día/mes/año (dd/mm/aa).

Los Patrones de Interacción resultan soluciones efectivas a problemas repetidos promoviendo la reutilización de los buenos diseños. Son una excelente alternativa para el desarrollo de interfaces WEB usables, en virtud de que se definen a partir de criterios de usabilidad. Sin embargo, una dificultad presente al momento, es que no se cuenta con una taxonomía de patrones aceptada universalmente.

II.2.3 Framework

Un framework puede ser definido como un conjunto de clases, y las colaboraciones que se establecen entre ellas para proporcionar un diseño abstracto para la solución de un conjunto de problemas [5].

Un Framework captura las decisiones de diseño comunes a un tipo de aplicaciones, estableciendo un modelo común a todas ellas, asignando responsabilidades y estableciendo colaboraciones entre las clases que forman el modelo. Pueden implementar uno o más patrones de diseño, y disponer también de un conjunto de patrones de interface o patrones de interacción. En el ámbito

de las aplicaciones Web, por lo general es implementado el patrón MVC (Modelo-Vista-Controlador) que separa en capas [5, 22]:

1) la lógica de negocios (El Modelo),

2) la presentación final al cliente de los datos procesados (La Vista), y

3) la capa encargada de recibir las entradas de usuario, para convocar a los métodos adecuados del modelo y de las vistas (El Controlador)

Los frameworks son generadores de aplicaciones que se relacionan directamente con un dominio específico, o una familia de problemas relacionados. En la actualidad su desarrollo está ganando una rápida aceptación debido a su capacidad para promover la reutilización de código del diseño y el código fuente.

II.2.3.1 Componentes básicos [13]

Básicamente un framework se compone de dos partes o conjunto de partes:

1. **Los puntos flexibles, o puntos calientes (hot-spots).** Son las clases o los métodos abstractos que deben ser implementados o puestos en ejecución.

2. **Los puntos estáticos, o puntos congelados (frozen-spots).** Son las características no mutables del framework y que tampoco pueden ser alteradas fácilmente.

Los Hot-spots constituyen la parte personalizable o configurable del framework, mediante la cual es posible añadir las últimas piezas para colocar el código específico y obtener una aplicación concreta.

Los Frozen-spots constituyen el núcleo o kernel del framework, y a diferencia de los hot-spots, se tratan de puntos inmutables puestos en ejecución dentro del framework que se encargan de llamar a uno o más puntos calientes proporcionados por el ejecutor.

II.2.3.2 Clasificación según su extensibilidad

Un framework puede clasificarse según su extensibilidad; pudiendo ser utilizado como una caja blanca o caja negra.

Los frameworks de caja blanca. También llamados frameworks con arquitectura de configuración-conducidos, la instanciación es posible a través de la creación de nuevas clases. Estas clases y el código pueden ser introducidas por herencia o composición. Uno debe programar el framework y entenderlo muy bien para producir un caso.

Los frameworks de caja negra. Producen instancias mediante el uso de scripts de configuración. Después de la configuración, una herramienta automática de instanciación crea las clases y el código fuente. También es posible utilizar asistentes para dirigir al usuario gradualmente a través del proceso de instanciación. Este tipo no requiere que el usuario conozca detalles internos del framework.

II.2.3.2 Desarrollo

Las tres etapas principales del desarrollo de un framework son: el análisis del dominio, diseño del framework, y su instanciación.

El análisis del dominio procura descubrir los requisitos del dominio y los posibles requerimientos futuros. Para completar los requerimientos sirven las experiencias previamente publicadas, los sistemas de software similares existentes, las experiencias personales, y los estándares considerados. Durante el análisis del dominio, los puntos calientes y los puntos congelados se establecen parcialmente.

La fase del diseño define las abstracciones de éste. Se modelan los puntos calientes y los puntos congelados (una buena alternativa sería mediante el uso de diagramas UML).

Finalmente, en la fase de "instanciación", los puntos calientes del framework son implementados generando el software del sistema. Es importante observar que los puntos calientes y congelados del framework serán siempre los mismos para cada una de las aplicaciones generadas.

Las fases del proceso de desarrollo de un framework pueden compararse con las fases tradicionales de diseño orientado a objeto, según se aprecia en la Figura II.3

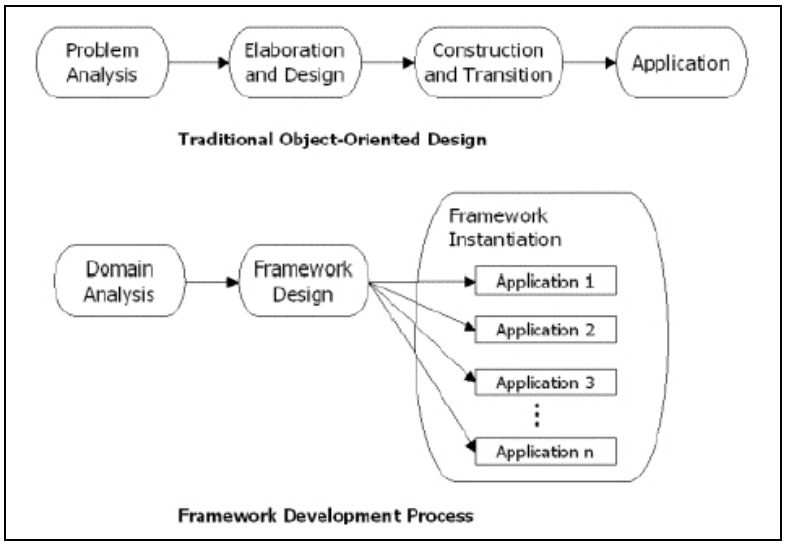


Figura II.3 - Fases de desarrollo de un framework [13]

En el desarrollo tradicional Orientado a Objeto, la fase de análisis del problema, estudia los requisitos de un solo problema. A diferencia, en el desarrollo de un framework se capturan los requisitos para un dominio entero. Además, el resultado final del desarrollo Orientado a Objeto tradicional es una aplicación completamente ejecutable, mientras que muchas aplicaciones pueden resultar a partir de la fase de "instanciación" en el desarrollo de un framework.

En general, los frameworks se desarrollan para la generación de aplicaciones dentro de un dominio en particular. Luego, una de las fases en el desarrollo de un framework, es el análisis del dominio en cuestión.

II.2.3.3 Beneficios

En general, los principales beneficios del uso de frameworks son [9]:

- **Modularidad y reducción de la complejidad.** La aplicación obtenida estará formada por subsistemas especializados en distintos aspectos del diseño.
- **Fortaleza al cambio.** Los módulos son evolutivos y fácilmente cambiables, conservando la arquitectura global de la aplicación.
- **Calidad de la Documentación.** La documentación adecuada promueve el uso correcto del framework y disminuye el esfuerzo necesario para el mantenimiento.

- **Estructura.** Establece una estructura sobre la cual las aplicaciones son construidas.

- **Distribución de funciones.** Permite un trabajo de desarrollo paralelo, ya que la solución puede desarrollarse como un conjunto de piezas independientes.

- **Eficiencia.** El desarrollador puede concentrarse en los requerimientos funcionales de la aplicación.

II.2.4 Conceptos Generales

A continuación, se presenta en términos generales algunos de los conceptos comunes a las aplicaciones Web. En particular, el framework propuesto en el presente trabajo, y por lo tanto todas las aplicaciones que se deriven del mismo, se basarán en las tecnologías que a continuación se detallan.

II.2.4.1 Lenguaje de Marca de Hipertexto, HTML

HTML es el acrónimo en inglés de Hyper Text Markup Language (lenguaje de marca de hipertexto), y como su nombre lo indica, es un lenguaje diseñado para estructurar información y presentarla en forma de hipertexto, que es el formato estándar de las páginas web.

HTML utiliza estas marcas o etiquetas, también conocidas como “tags”, para determinar la forma en que la información (textos, imágenes u otros elementos) son presentados a través de un navegador.

II.2.4.2 Lenguaje de Marcado Extensible, XML [8]

XML al igual que HTML, consiste de un lenguaje de marcas o “tags”, pero con algunas diferencias importantes:

- Mientras HTML se ocupa del aspecto de los datos (color, tamaño, tipo de fuente, etc.), XML se preocupa por su significado o semántica.

- Mediante XML es posible definir nuestro propio conjunto de etiquetas, o nuestro propio lenguaje, para describir con precisión que es lo que se desea.

- Toda la información almacenada en un documento XML resulta independiente de su representación. Luego, mediante páginas de estilos XSL, o por medio de otros métodos de transformación, es posible producir múltiples salidas.

En nuestro caso, se utiliza este estándar para la especificación de los distintos patrones de interacción del framework propuesto.

II.2.4.3 Document Object Model, DOM [3, 6]

El Modelo de Objeto de Documento, es un estándar para acceder y manipular los documentos XML y HTML en los navegadores, editores y otro tipo de aplicaciones.

II.2.4.4 AJAX, Asynchronous JavaScript And XML [3, 6]

Se trata de una técnica de desarrollo web para la creación de aplicaciones interactivas que combina tres tecnologías existentes:

1. **HTML** para presentar la información,
2. **Document Object Model (DOM) y JavaScript**, para interactuar dinámicamente con los datos, y
3. **XML**, para el intercambio de datos asincrónicamente con un servidor web.

En un esquema tradicional, los usuarios interactúan mediante páginas, en dónde cada acción, implica el envío de los datos al servidor para ser procesados y posteriormente ser devueltos al cliente. La principal desventaja es el desperdicio de ancho de banda en la comunicación entre el cliente y el servidor. El usuario siempre espera a que toda la página con la que interactúa sea recargada. (Figura II.4)

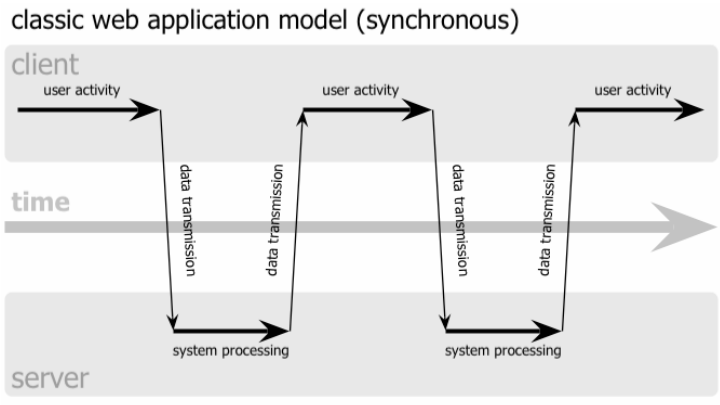


Figura II.4 - Modelo de aplicación Web Tradicional [6]

A diferencia del esquema tradicional, las aplicaciones AJAX envían al servidor sólo peticiones puntuales en segundo plano, sin necesidad de recargar una página completa sino porciones de ésta, lo cual redunda en una mayor interacción y rapidez en los tiempos de respuesta gracias a la reducción de la información intercambiada entre el servidor y el cliente, y a que una parte del proceso de la información es realizada en el propio cliente, liberando al servidor de ese trabajo. (Figura II.5)

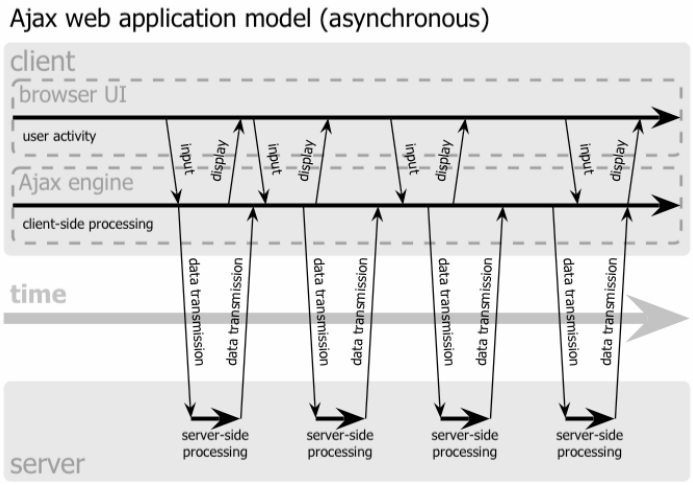


Figura II.5 - Modelo de aplicación Web basada en AJAX [6]

II.2.4.5 PHP [19, 23, 2]

La infraestructura actual de la Web, sirve hoy para mucho más que ofrecer páginas estáticas, existiendo una infinidad de tecnologías que facilitan la creación de contenidos dinámicos. Una de esas soluciones, y es probable que la más extendida sea PHP. Mediante PHP, es posible crear aplicaciones Web completas que se ejecutan del lado del servidor, y que pueden ser accedidas desde cualquier cliente o navegador. Desde 1994 hasta ahora, PHP ha evolucionado de

ser un conjunto de utilidades de uso personal de su original creador, hasta convertirse en un lenguaje orientado a objetos puro, en donde el núcleo del lenguaje (desarrollado por la empresa Zend Technologies), introduce un modelo de Orientación a Objetos similar al de Java.

Desde 1994 a esta parte, PHP ha pasado de un ámbito personal, a convertirse en la solución de creación de páginas web a escala mundial, con un uso estimado que supera los dieciséis millones de dominios.

Es por su flexibilidad, su característica multiplataforma, y su uso ampliamente extendido que se ha elegido a PHP como lenguaje para implementar el framework propuesto.

II.2.4.6 MySQL [2]

MySQL es uno de los RDBMS (Sistema de administración de base de datos relacionales) más populares, ya que lleva mucho tiempo utilizándose en sistemas operativos como Linux, y que dispone además de una licencia de tipo Dual, que permite tanto su uso libre, como así también su adquisición comercial.

En su versión actual, MySQL incorpora todas las características de un RDBMS completo, con soporte a las transacciones, integridad referencial, procedimientos almacenados, cron jobs (o tareas programadas), y a diferencia de otros Motores de Bases de datos, MySQL está disponible prácticamente para todos los sistemas operativos actuales, desde Windows, hasta Mac OS, OS/2, FreeBSD, Unix, Solaris, Unixware, así como para plataformas con procesadores de distintas familias de 32 y 64 bits, siendo incluso capaz de aprovechar configuraciones multiproceso.

Es por su potencia, su uso extendido, y su característica de software libre, que se ha elegido MySQL como motor de base de datos para dar soporte a la gestión y administración de contenidos.

II.3. Marco Metodológico

II.3.1 UML [11, 12, 20]

UML (Unified Modeling Language) es un lenguaje para modelar, construir y documentar los elementos que conforman un sistema software orientado a objetos. Se ha convertido en el estándar de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa: Booch, OMT y OOSE. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos [4].

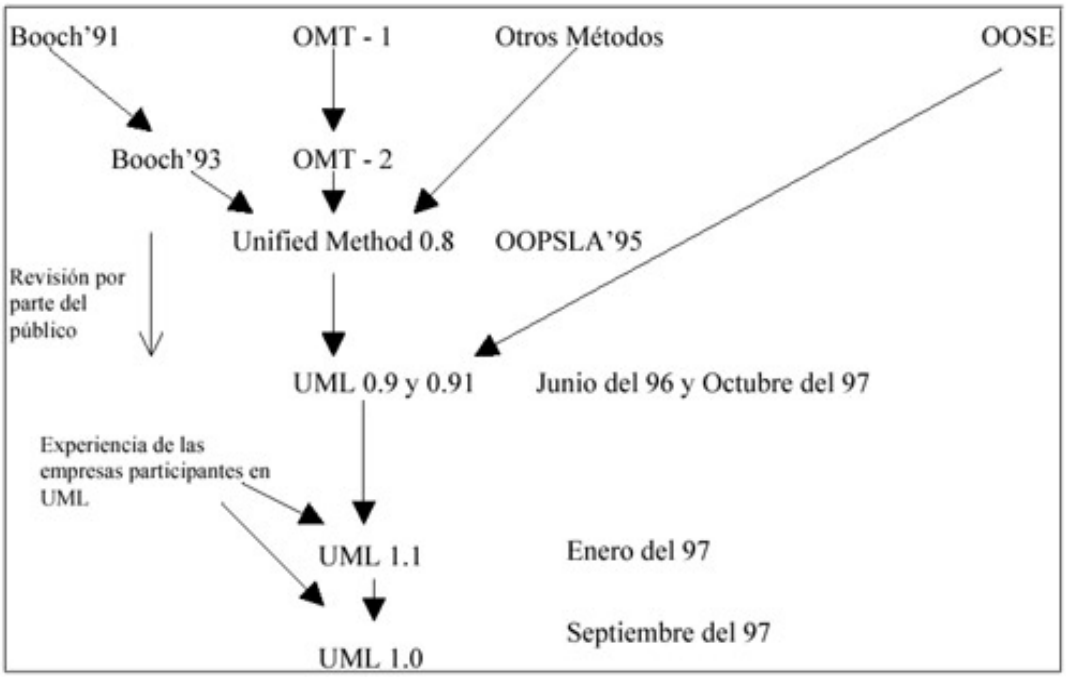


Figura II.6 – Evolución de UML [4]

La notación UML permite modelar y representar gráficamente los principales conceptos de la Orientación a Objetos. Cada modelo UML proporciona una visión particular de algún aspecto del sistema [4]:

- **Diagramas de Estructura Estática.** Engloba tanto al Modelo Conceptual de la fase de Análisis como al Diagrama de Clases de la fase de Diseño. Ambos son distintos conceptualmente, mientras el primero modela elementos del dominio el segundo presenta los elementos de la solución software. Sin embargo, ambos comparten la misma notación para los elementos que los forman (clases y objetos) y las relaciones que existen entre los mismos (asociaciones).

- **Diagramas de Casos de Uso.** Muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

- **Diagramas de Secuencia.** Muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo.

- **Diagramas de Colaboración.** Muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

- **Diagrama de Estados.** Muestra la secuencia de estados por los que pasa un caso de uso o un objeto a lo largo de su vida, indicando qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

A continuación se describe en detalle, la sintaxis correspondiente a los diagramas de clases y a los diagramas de secuencia o interacción. Las mismas son utilizadas para modelar los diferentes patrones de diseño, y representar el patrón de arquitectura MVC propuesto en presente trabajo.

II.3.1.1 Diagramas de Clases

Con el nombre de *Diagrama de Clases,* se hace referencia a los diagramas de la fase de diseño para presentar las clases, y las relaciones entre las mismas.

II.3.1.1.1 Clases

Una clase se representa mediante una caja dividida en tres partes: En la superior se muestra el nombre de la clase, en el medio las operaciones principales y en la inferior las variables de instancia. Una clase puede representarse de forma esquemática (plegada), con los detalles como atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase.

Figura II.7 - Clases Abstractas y concretas

II.3.1.1.2 Asociaciones

Las asociaciones entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación:

- **Nombre de la asociación y dirección.** El nombre de la asociación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir un pequeño triángulo negro sólido que indique la dirección en la cual leer el nombre de la asociación.

- **Multiplicidad.** La multiplicidad es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase. Puede expresarse de las siguientes formas:

- ✓ Con un número fijo: 1
- ✓ Con un intervalo de valores: 2..5
- ✓ Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, 2..* significa 2 o más
- ✓ Con una combinación de elementos como los anteriores separados por comas: 1,3..5, 7, 15..*
- ✓ Con un asterisco: * . En este caso indica que puede tomar cualquier valor (cero o más)

- **Roles.** Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol. Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

- **Agregación.** El símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el “todo”.

- **Asociaciones n-arias.** En el caso de una asociación en la que participan más de dos clases, las clases se unen con una línea a un diamante central.

- **Navegabilidad.** En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible "navegar" desde el objeto de la clase origen hasta el objeto de la clase destino. Se trata de un concepto de diseño, que indica que un objeto de la clase origen conoce al (los) objeto(s) de la clase destino, y por tanto puede llamar a alguna de sus operaciones.

II.3.1.1.3 Herencia

La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o clase “padre”.

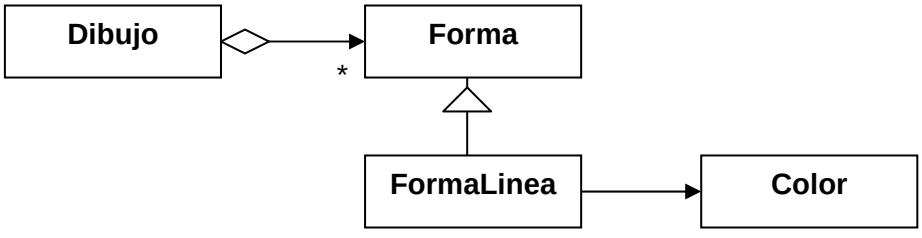


Figura II.8 - Ejemplo de diferentes relaciones entre clases [5]

En la figura anterior se muestran diferentes relaciones entre clases:

Un triángulo que conecta a la subclase (FormaLinea) con su clase padre (Forma), denota la herencia de clases. Una referencia a un objeto que representa una relación de agregación o parte-todo, se indica mediante una flecha con un rombo en su base. La flecha apunta a la clase del agregado (Forma). Una flecha sin rombo denota una asociación (por ejemplo, FormaLinea tiene una referencia a un objeto Color que puede ser compartido con otras formas). El asterisco al final de la flecha de agregación quiere decir que Dibujo tiene múltiples objetos de tipo forma.

II.3.1.2 Diagramas de secuencia

Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. Muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo.

El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo. Un rectángulo vertical indica que un objeto está activo, es decir que está procesando una petición. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) en el margen izquierdo, o bien junto a las transiciones.

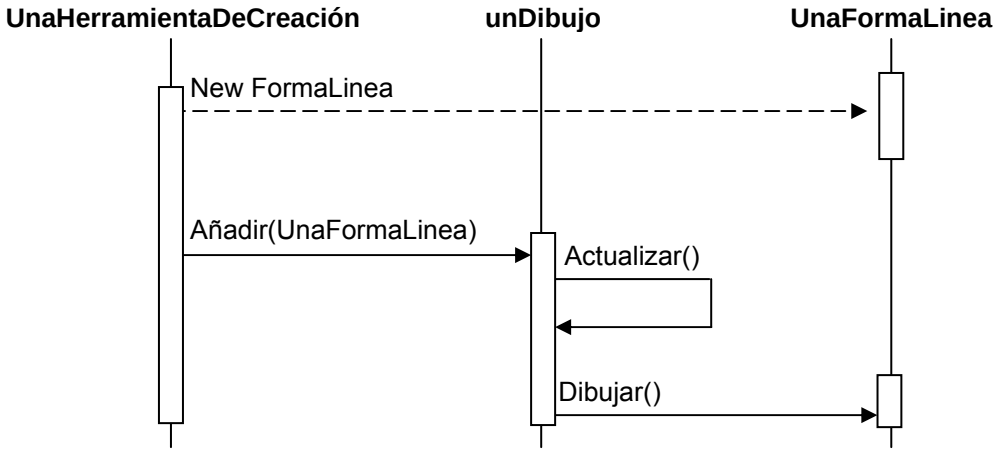


Figura II.9 - Ejemplo de Diagrama de secuencia [5]

En la Figura II.9 se observa que la primer petición procede de *UnaHerramientadeCreación* para crear *UnaFormaLinea* (toda petición de creación

de objeto se indica con línea punteada). A continuación se añade *UnaFormaLinea* a *UnDibujo*, lo que hace que *UnDibujo* se envíe a sí mismo una petición de Actualizar. *UnDibujo* también envía una petición de dibujar a *UnaFormaLinea* como parte de la operación Actualizar.

II.3.2 COCOMO y COCOMO II

El modelo COCOMO fue desarrollado por Barry .W Boehm a finales de la década de 1970 y comienzos de 1980, exponiéndolo detalladamente en su libro "Software Engieneering Economis", publicado por Prentice-Hall en 1981.

COCOMO es una jerarquía de modelos de estimación de costos de software que incluye los submodelos básico, intermedio y avanzado, de acuerdo con el tamaño del proyecto respectivamente [7].

El *modelo básico* calcula el esfuerzo y el costo de desarrollo en función del tamaño del programa estimado en miles de líneas de código.

El *modelo intermedio* calcula el esfuerzo de desarrollo en función del tamaño del programa y un conjunto de conductores del costo que incluyen la evaluación subjetiva del producto, del hardware, del personal y de atributos del proyecto.

El *modelo avanzado* incorpora las características del modelo intermedio, y además lleva a cabo una evaluación del impacto de los conductores del costo en cada fase (análisis, desarrollo, etc.) del proceso.

Los modelos COCOMO tradicional, están definidos para tres tipos de proyecto software:

- **Orgánico.** Proyectos pequeños y sencillos, con equipos chicos con experiencia en la aplicación, y con requisitos poco rígidos.
- **Semiacoplado**. Proyectos de tamaño y complejidad intermedios, con equipos con variados niveles de experiencia, y con requisitos poco o medio rígidos.
- **Empotrado.** Incluye proyectos que deben ser desarrollados con un conjunto de requisitos muy estrictos.

A continuación se presentan las ecuaciones y coeficientes, para los modelos antes mencionados.

1) Modelo COCOMO Básico [7].

$$\mathbf{E} = ab\ (\mathbf{KLDC})^{bb}$$

$$\mathbf{D} = cb\ (\mathbf{E})^{db}$$

$$\mathbf{P} = \mathbf{E} / \mathbf{D}$$

Dónde **E** es el esfuerzo aplicado en persona-mes, **D** el tiempo de desarrollo en meses, **KLDC** el número de líneas (en miles) estimadas para el proyecto, y **P** el número de personas necesarias. Los valores de los coeficientes **ab**, **bb**, **cb** y **db** dependen del tipo de proyecto, y son los que se describen en la siguiente tabla:

Modelo COCOMO Básico				
Proyecto Software	**ab**	**bb**	**cb**	**db**
Orgánico	2,4	1,05	2,5	0,38
Semiacoplado	3,0	1,12	2,5	0,35
Empotrado	3,6	1,20	2,5	0,32

Figura II.10 - Coeficientes COCOMO básico [7]

Si bien el modelo no estima directamente el costo, brinda mediante ecuaciones tres factores influyentes en el mismo: el esfuerzo (expresado en meses/hombre), la duración del proyecto y la cantidad de personas necesarias para la ejecución del mismo

2) Modelo COCOMO Intermedio [7].

$$\mathbf{E} = ai\ \mathbf{KLDC}^{\,bi} \times \mathbf{FAE}$$

Dónde **KLDC** el número de líneas (en miles) estimadas para el proyecto y **FAE** el factor de ajuste de esfuerzo. **FAE**: $\Pi i = 1..15$ Driver $_i$

Modelo COCOMO Intermedio		
Proyecto Software	**ai**	**bi**
Orgánico	3,2	1,05
Semiacoplado	3,0	1,12
Empotrado	2,8	1,20

Figura II.11 - Coeficientes COCOMO intermedio [7]

El modelo básico se amplía con un conjunto de atributos conductores del costo que pueden agruparse en cuatro categorías principales: a) atributos del producto, b) del hardware, c) del personal y d) del proyecto.

Esto resulta en 15 atributos que son valorados en una escala de 6 puntos desde "muy bajo" a "extra alto".

Atributos	Valor					
	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Atributos del producto						
Fiabilidad	,75	,88	1,00	1,15	1,40	
Tamaño base de datos		,94	1,00	1,08	1,16	
Complejidad	,70	,85	1,00	1,15	1,30	1,65
Atributos del computador						
Restricciones de tiempo de ejecución			1,00	1,11	1,30	1,66
Restricciones de memoria virtual			1,00	1,06	1,21	1,56
Volatilidad de la máquina virtual		,87	1,00	1,15	1,30	
Tiempo de respuesta		,87	1,00	1,07	1,15	
Atributos del personal						
Capacidad de análisis	1,46	1,19	1,00	,86	,71	
Experiencia en la aplicación	1,29	1,13	1,00	,91	,82	
Calidad de los programadores	1,42	1,17	1,00	,86	,70	
Experiencia en la máquina virtual	1,21	1,10	1,00	,90		
Experiencia en el lenguaje	1,14	1,07	1,00	,95		
Atributos del proyecto						
Técnicas actualizadas de programación	1,24	1,10	1,00	,91	,82	
Utilización de herramientas software	1,24	1,10	1,00	,91	,83	
Restricciones de tiempo de desarrollo	1,23	1,08	1,00	1,04	1,10	

Figura II.12 - Atributos conductores de costo (modelo intermedio) [7]

El modelo original COCOMO en general refleja las prácticas de desarrollo de software de la década de los 70. Luego de casi una década y media, las técnicas de desarrollo cambiaron drásticamente, con un énfasis creciente en la reutilización de software y la construcción de sistemas utilizando componentes software a medida.

II.3.2.1 Puntos de Objeto (PO) [16]

Uno de los problemas del COCOMO original es su dependencia con la Cantidad de Líneas de Código (KLDC) como una variable clave. COCOMO II (Boehm 96) reconoce que es difícil estimar en base a KLDC en etapas tempranas. Es por eso que COCOMO II se basa en los puntos objeto.

El punto objeto es un medida indirecta del software que se calcula teniendo en cuenta el total de:

- Pantallas o interfaces de usuario

- Informes
- Componentes necesarios para construir la aplicación

Cada instancia de objeto se clasifica en un nivel de complejidad, según la siguientes tablas:

Pantallas			
Nro. de Vistas	**Fuente de tabla de datos**		
	< 4	< 8	> 8
< 3	**Básico**	**Básico**	**Intermedio**
3 – 7	**Básico**	**Intermedio**	**Avanzado**
> 8	**Intermedio**	**Avanzado**	**Avanzado**

Informes			
Nro. de Secciones	**Fuente de tabla de datos**		
	< 4	< 8	> 8
0 ó 1	**Básico**	**Básico**	**Intermedio**
2 ó 3	**Básico**	**Intermedio**	**Avanzado**
> 4	**Intermedio**	**Avanzado**	**Avanzado**

Cada nivel de complejidad tiene un peso asignado según el tipo de objeto, como se muestra en la siguiente tabla:

Tipo de Objeto	**Peso de la Complejidad**		
	Básico	**Intermedio**	**Avanzado**
Pantalla	1	2	3
Informe	2	5	8
Componentes	-	-	10

El total de puntos de objeto se determina con la sumatoria de multiplicar el número original de instancias de objeto, por el factor de peso asignado. **Total PO =** $\sum$ (nro. instancias x peso)

Cuando se aplica sobre el desarrollo mediante componentes o para la reutilización en general, se estima el porcentaje de reutilización, y se calculan los Puntos de Objeto Nuevos (NOP).

NOP = PO x [(100 - %reutilización) / 100]

Por último se calcula el Esfuerzo estimado, expresado en Meses-Persona como:

MM = NOP / PROD

Dónde **PROD** es la proporción de productividad basado en la experiencia del desarrollador y la madurez del entorno de desarrollo, según la siguiente tabla:

Experiencia/capacidad del desarrollador (ECD)	Muy Baja	Baja	Normal	Alta	Muy Alta
Madurez/capacidad del entorno (MCD)	Muy Baja	Baja	Normal	Alta	Muy Alta
PROD	4	7	13	25	50

Figura II.13 - Proporción de Productividad COCOMO II [16]

II.4. Marco Empírico

El universo de estudio del presente trabajo se limita al dominio de las aplicaciones Web para la gestión de contenidos, en particular contenidos de tipo periodísticos. Por lo tanto, se toma como base de análisis :

- Frameworks comerciales orientados a diferentes propósitos.
- Diferentes patrones de interacción para entornos Web
- Una aplicación Web para la gestión de contenidos periodísticos desarrollada a medida y aplicada a un conocido diario digital local.
- La experiencia de un grupo de periodistas que vienen desempeñándose desde hace un tiempo, en el ámbito de la publicación online de diarios digitales en nuestro medio.

En base a los resultados obtenidos del análisis, se pretende volcar los conocimientos adquiridos, en el desarrollo de un framework basado en patrones de diseño y patrones de interacción que permita extender en forma rápida y sencilla aplicaciones web para la administración de este tipo de contenidos.

El framework obtenido deberá permitir:

- Extender aplicaciones a partir de un conjunto claro de especificaciones.
- Desarrollar aplicaciones fáciles de mantener.
- Generar aplicaciones basadas en el patrón de arquitectura MVC.
- Lograr aplicaciones con interfaces claras y fáciles de utilizar, con toma de decisiones del tipo Si/No, Aceptar/Cancelar.
- Permitir aplicaciones con interfaces para la gestión de código HTML transparentes por parte del usuario final.

Capítulo 3

Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web

Desarrollo de un Framework basado en Patrones

III.1. Introducción

Como ya se indicó, las tres etapas principales del desarrollo de un framework son: el análisis del dominio, diseño del framework, e instanciación.

En el análisis del dominio se determinan los requerimientos del dominio y los posibles requerimientos futuros. En la fase del diseño se definen las abstracciones de éste, y se modelan los puntos calientes y los puntos congelados. Por último en la fase de instanciación, los puntos calientes son implementados, generando las aplicaciones extendidas del framework.

III.2 Análisis del Dominio

En base al análisis del dominio especificado en la sección II.4, se establecen un conjunto requerimientos relacionados al framework, y al conjunto de aplicaciones que se extenderán del mismo:

1. **Implementar el patrón de arquitectura MVC.** El patrón de arquitectura MVC permite un menor acoplamiento, una mayor cohesión y una mayor claridad en el diseño facilitando el posterior mantenimiento. (Sección II.2.1.4)

2. **Proponer un conjunto claro de instrucciones para la instanciación del framework.** Este punto hace referencia a los puntos calientes que deben ser puestos en ejecución para la generación de una aplicación específica. (Sección II.2.3.1)

3. **Generar aplicaciones con soporte AJAX.**

4. **Generar aplicaciones multiplataforma.** Las aplicaciones extendidas deben poder ser ejecutadas sobre diferentes plataformas, con lo cual se elige a PHP (como lenguaje de desarrollo) y MySQL (como motor de base de datos) como las tecnologías adecuadas.

5. **Extender aplicaciones con interfaces uniformes y fáciles de usar.**

6. **Permitir generar código HTML de forma transparente.**

7. **Mejorar los tiempos de respuesta de las aplicaciones generadas.**

III.3. Diseño del Framework

El objetivo del presente trabajo es obtener un framework basado en patrones de diseño y patrones de interacción. A continuación se definen:

a) un conjunto de patrones de interacción (resultado final de las vistas dentro del patrón de arquitectura MVC), y

b) el patrón de arquitectura MVC.

III.3.1. Definición de los patrones de interacción

El objetivo de contar con patrones de interacción, es poder dotar a las aplicaciones, de interfaces uniformes, flexibles y fáciles de aprender, tratando de mejorar la usabilidad en la interacción con los usuarios.

Los patrones propuestos, han sido definidos en base a:

- La experiencia en el desarrollo de aplicaciones orientadas a la Web
- El análisis de diferentes frameworks libres y de uso comercial (Sección I.3)
- El uso de componentes y librerías de uso libre disponibles en la Web
- Diferentes guías de usabilidad disponibles
- Las necesidades funcionales por parte de los usuarios

A continuación se presentan los diferentes patrones propuestos, clasificados en simples y compuestos. Los patrones simples, incorporan funcionalidad en combinación, únicamente dentro de los patrones de interacción complejos, que en definitiva conforman un patrón de interface de usuario completo.

III.3.1.1. Patrones de interacción simples

Los patrones simples, permiten navegar a través de un conjunto de registros, o bien, proporcionan un mecanismo para el ingreso de diferentes tipos datos, ayudando a los usuarios a prevenir errores. En este ultimo caso, se distinguen los componentes "*DHTML Calendar"*, y "*FCKEditor HTML"*, ambos de uso libre y desarrollados por Mihai Bazon y Frederico Caldeira Knabben respectivamente.

1) Campo Lista de Selección DBComboBox

Nombre. Campo Lista de selección DBComboBox.

Autor. Martín Murillo

Problema. Dada una tabla de una base de datos (por ejemplo provincias), el usuario desea seleccionar un valor descriptivo sin tener que preocuparse del valor numérico que lo identifica.

Principio de Usabilidad. Guiar al usuario y prevenir errores

Contexto. Todos los sistemas que requieran que el usuario introduzca valores por medio de una lista de selección de valores.

Fuerzas. Las listas con valores descriptivos en una base de datos se encuentran por lo general codificados.

Solución. Permitir que el usuario elija un valor descriptivo como en el mundo real.

Ejemplo. Seleccionar una provincia de una lista de provincias disponibles, sin preocuparse del valor numérico que identifica la provincia seleccionada.

2) Campo de Selección Prompt

Nombre. Campo de Selección Prompt.

Autor. Martín Murillo

Problema. Permite disparar una ventana de búsqueda que retorna un valor seleccionado.

Principio de Usabilidad. Guiar al usuario y prevenir errores

Contexto. Todos los sistemas que requieran que el usuario introduzca valores por medio de una lista de selección con un número relativamente grande de registros.

Fuerzas. Las listas con valores descriptivos en una base de datos se encuentran por lo general codificados.

Solución. Permitir que el usuario elija un valor descriptivo como en el mundo real.

Ejemplo. Seleccionar una provincia de una lista de provincias disponibles, sin preocuparse del valor numérico que identifica la provincia seleccionada.

3) DHTML Calendar

Nombre. DHTML Calendar.

Autor. Mihai Bazon para Dynarch.com

Problema. El usuario desea introducir datos de fechas y no desea preocuparse por la sintaxis del dato.

Principio de Usabilidad. Guiar al usuario y prevenir errores

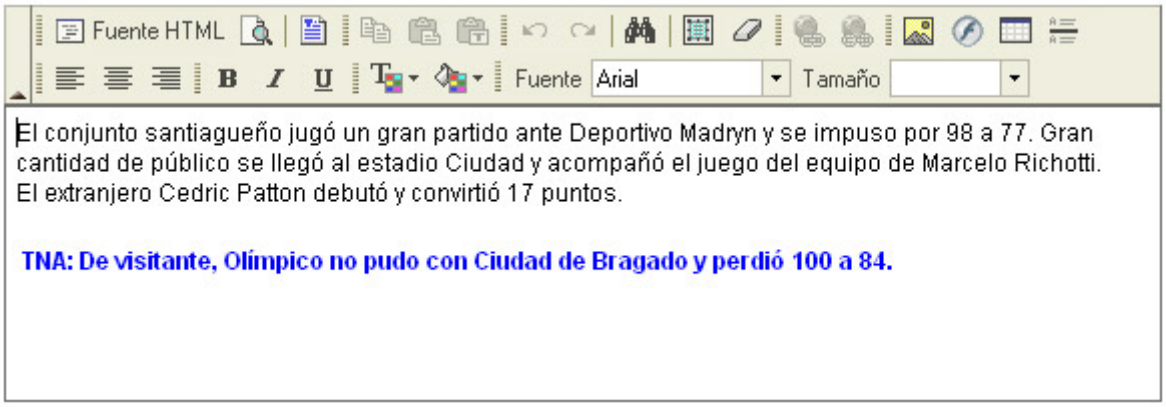
Contexto. Todos los sistemas que requieran que el usuario introduzca fechas

Fuerzas. Los datos de fechas tienen múltiples sintaxis.

Solución. Permitir que el usuario elija la fecha de un calendario como en el mundo real.

Ejemplo. Para realizar la búsqueda de algún registro creado a partir de una fecha, el calendario ayudará a llenar los campos de fecha con el formato correcto año-mes-día (aa-mm-dd).

4) FCKEditor HTML



Nombre. FCKEditor HTML.

Autor. Frederico Caldeira Knabben (fredck@fckeditor.net)

Problema. El usuario desea formatear texto, incorporar imágenes, archivos multimedia, etc. El objetivo es poder generar código HTML en forma transparente.

Principio de Usabilidad. Guiar al usuario y prevenir errores

Contexto. En sistemas Web para la gestión de contenidos en ocasiones es necesario poder introducir código HTML.

Fuerzas. El código HTML puede resultar complejo para la mayoría de los usuarios inexpertos.

Solución. Permitir a un usuario inexperto generar código HTML.

Ejemplo. Un texto enriquecido mediante formateo de color, tamaño de fuente, incorporación de imágenes, etc..

5) Grilla de Navegación / Edición

Nombre. Grilla de Navegación / Edición

Autor. Martín Murillo

Problema. El usuario desea recorrer un conjunto de registros, y realizar algún tipo de acción (editar, borrar o llamar a una vista) asociada a un registro específico, o ingresar un nuevo registro.

Principio de Usabilidad. Permite un mecanismo de navegación sencillo.

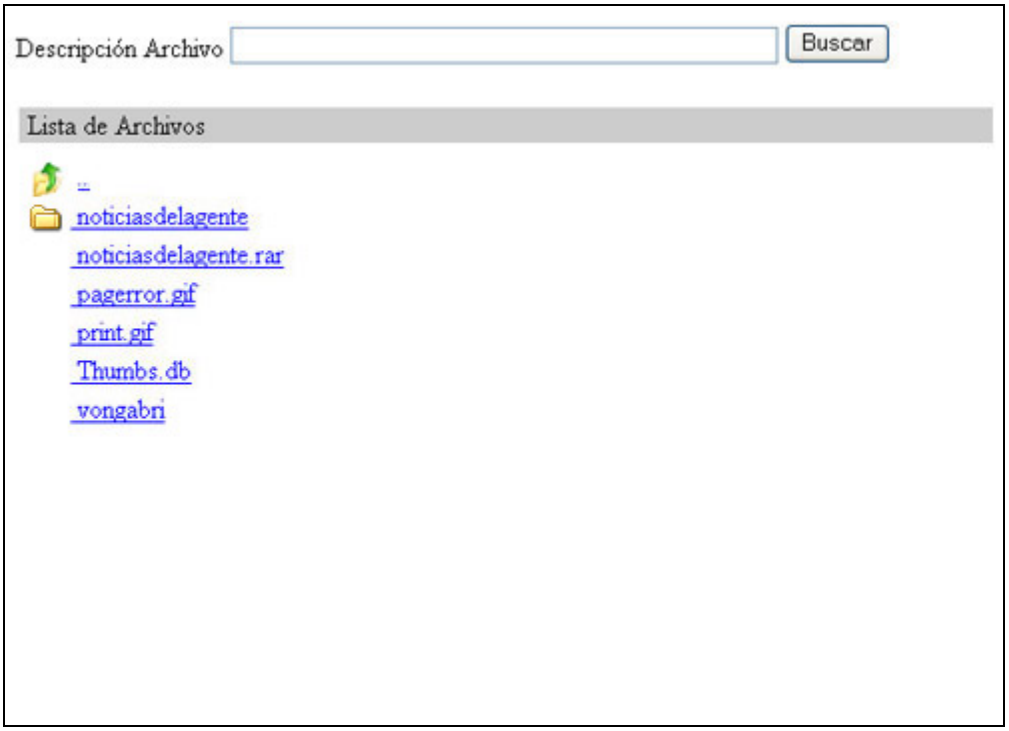
Contexto. Permite un mecanismo de navegación sencillo.

Fuerzas. Proporciona un punto de acceso en común para convocar a las diferentes acciones asociadas a un registro.

Solución. Permite recorrer un conjunto de registros, y convocar a las diferentes acciones de edición asociadas a un registro específico.

Ejemplo. Dada base de datos, permite recorrer los registros de una tabla.

6) Explorador de Directorio



Nombre. Pantalla Explorador de Directorio.

Autor. Martín Murillo

Problema. El usuario necesita navegar, filtrar datos y retornar el nombre de un archivo dentro de un directorio determinado.

Principio de Usabilidad. Permite recorrer y filtrar datos dentro de un directorio.

Contexto. Dado un directorio específico, proporciona un mecanismo de filtro y navegación.

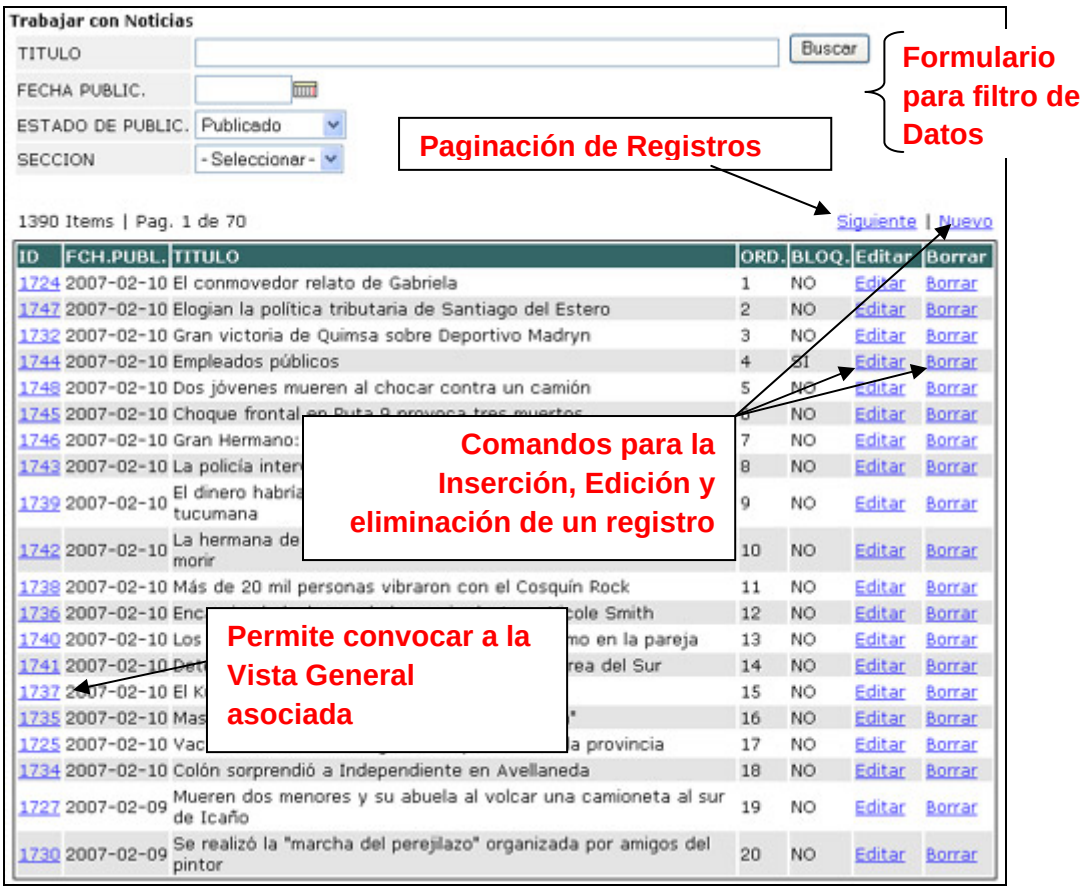
Fuerzas. Proporciona un entorno de navegación de datos fácil de utilizar.

Solución. Proporciona un mecanismo para el filtro de datos por medio de un formulario de búsqueda, y permite retornar el nombre de un archivo dentro de un directorio específico. La pantalla Explorador de Directorio es invocada por el Campo de Selección Prompt.

III.3.1.2. Patrones de interacción compuestos

Los patrones de interacción compuestos combinan en su estructura uno o más patrones de interacción simples.

1) Pantalla Maestro de Registros. Este patrón permite combinar en un formulario de filtro de datos, los diferentes patrones simples para el ingreso de datos (DBComboBox, Selección Prompt), y además incorpora el patrón Grilla de Navegación, para el maestro de registros.



Nombre. Pantalla Maestro de Registros.

Autor. Martín Murillo

Problema. El usuario necesita navegar, filtrar datos y convocar a los procesos de edición y relación de contenidos para un registro deseado.

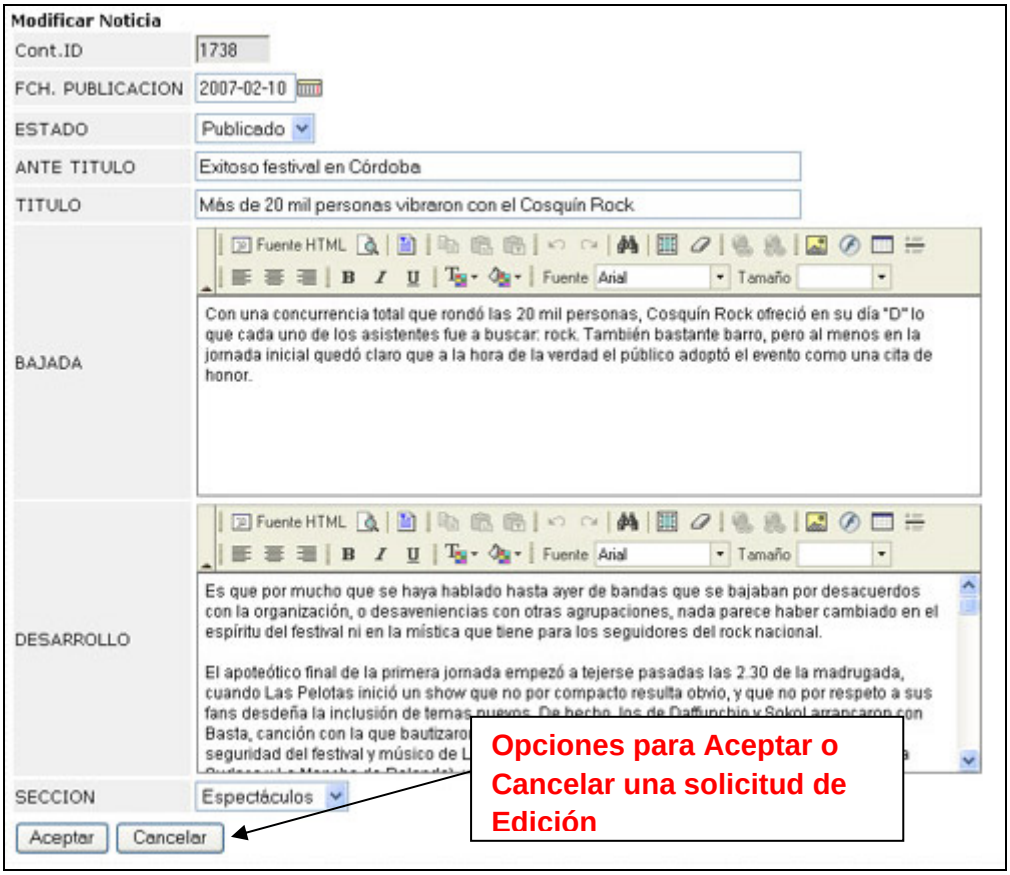
Principio de Usabilidad. Permite recorrer y filtrar datos, pero no permite modificarlos.

Contexto. Dada una Base de Datos, proporciona un mecanismo de trabajo y navegación fácil de comprender, para una tabla o un conjunto de tablas asociadas.

Fuerzas. Proporciona un entorno de navegación de datos fácil de utilizar.

Solución. Proporciona un mecanismo para el filtro y navegación de datos, y permite convocar a los procesos de alta, baja, modificación y gestión de contenidos relacionados a un registro de datos específico.

2) Formularios de Edición. Permite incorporar y combinar los diferentes patrones simples para el ingreso de datos. En la figura se distinguen los patrones FCKEditor HTML y DHTML Calendar.



Nombre. Formulario de Edición.

Autor. Martín Murillo

Problema. El usuario necesita Editar/Crear/Borrar un registro.

Principio de Usabilidad. Son los únicos que permiten la modificación de datos. Incorporan una acción de aceptación o cancelación explícitas. En caso de cancelar la acción se retorna a la pantalla desde dónde fue convocada.

Contexto. Genera pantallas de edición, con los campos y tipos de datos de entrada adecuados.

Fuerzas. Permiten la validación de los datos ingresados.

Solución. Permite el alta, baja o modificación de un registro específico.

Ejemplo. Dada una tabla de una base de datos, permite el alta o modificación de un registro específico.

3) Pantalla Vista General. En la solapa *"General"*, se combinan los diferentes patrones simples para el ingreso de datos (pero de sólo lectura), e incorpora el patrón Grilla de Navegación para cada una de las vistas restantes asociadas.

Nombre. Pantalla Vista General.

Autor. Martín Murillo

Problema. El usuario necesita relacionar los diferentes contenidos asociados a un registro.

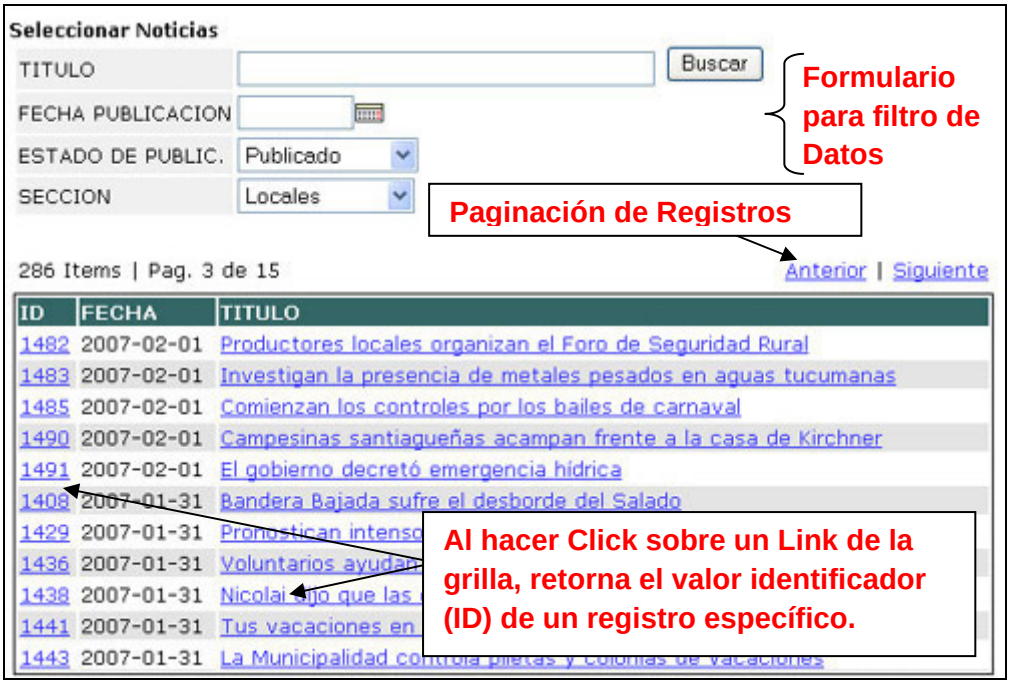
Principio de Usabilidad. Permite recorrer y filtrar datos relacionados, pero no permite modificarlos. Proporciona un conjunto de N vistas asociadas a un registro específico.

Contexto. Dada una Base de Datos, proporciona un mecanismo de trabajo y navegación fácil de comprender para un conjunto de tablas asociadas.

Fuerzas. Proporciona un entorno de navegación de las diferentes vistas fácil de utilizar.

Solución. Proporciona un mecanismo para modelar las relaciones 1:N o N:N de una base de datos relacional.

4) Selector Prompt. En el formulario de filtro de datos, combina los diferentes patrones simples para el ingreso de datos, e incorpora además el patrón *Grilla de Navegación*, para el maestro de registros.



Nombre. Pantalla Selector Prompt.

Autor. Martín Murillo

Problema. El usuario necesita navegar, filtrar datos y retornar el valor identificador de un registro deseado.

Principio de Usabilidad. Permite recorrer y filtrar datos, pero no permite modificarlos.

Contexto. Dada una Base de Datos, proporciona un mecanismo de filtro y navegación fácil de comprender, para una tabla o un conjunto de tablas asociadas.

Fuerzas. Proporciona un entorno de navegación de datos fácil de utilizar.

Solución. Proporciona un mecanismo para el filtro de datos por medio de un formulario de búsqueda, y retornar el valor identificador de un registro específico. La pantalla Selector Prompt es invocada por el Campo de Selección Prompt.

III.3.1.3. Modelo de navegación

Los patrones de interacción definen el lenguaje con el que los usuarios interactúan y se comunican con las aplicaciones, por lo tanto cuando más simple, consistente y claro sea el lenguaje, más fácil de usar será el sistema.

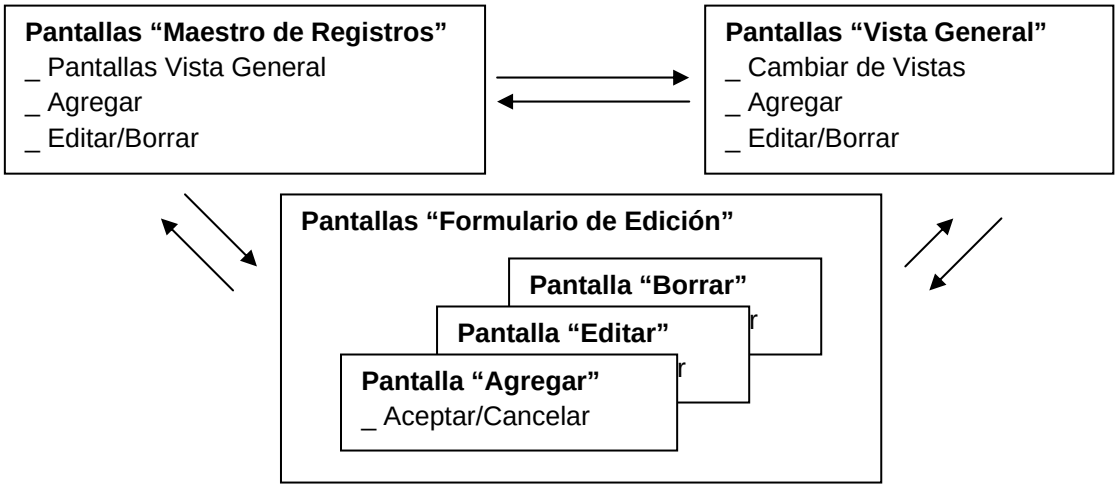


Figura III.1 – Modelo de navegación

Como se observa en la Figura III.1, las aplicaciones extendidas del framework propuesto tienen como punto de inicio una pantalla "Maestro de Registros", que proporciona un medio para navegar por los mismos.

- Una pantalla "Maestro de Registros", permite paginar y filtrar registros, convocar a una "Vista General", y convocar en modo Inserción, Edición o Eliminación a una pantalla "Formulario".

- Una pantalla “Vista General”, permite navegar por los conjuntos de registros asociados a un registro dado, y convocar en modo Inserción, Edición o Eliminación a una pantalla “Formulario”.

- Una pantalla “Formulario“ permite Aceptar o Cancelar una acción de actualización, retornando en ambos casos al punto anterior desde dónde fue convocada.

- Toda acción Cancelar/Aceptar desde una pantalla formulario, o una acción Volver desde una pantalla “Vista General”, retorna al punto anterior desde dónde fueron convocadas.

III.3.2. Definición del patrón MVC

Los patrones de interacción son el resultado de las “Vistas” del patrón de arquitectura MVC. Las vistas se encargan de generar los patrones de interface a partir de un conjunto de especificaciones XML, que son interpretadas en el lado del cliente por su correspondiente estilo XSL.

Cada patrón de interacción, tiene asociado un conjunto de comandos o controles JavaScript, del lado del cliente, encargados de ejecutar las diferentes peticiones que luego se interpretan del lado del servidor.

El conjunto de comandos o controles de peticiones JavaScript, y los mecanismos de interpretación y control de dichas peticiones del lado del servidor, constituyen la capa “Control” del patrón MVC.

Una petición al servidor puede tratarse de una actualización o una simple consulta. Cuando el controlador interpreta el comando solicitado, éste delega la responsabilidad de su ejecución a la capa “Vista” o “Modelo”. Cada vez que se produce un cambio en el “Modelo”, éste informa a las capas “Vista” y “Controlador”, quienes reflejan el nuevo estado.

En la Figura III.2 se muestra el patrón MVC del framework propuesto, mediante un diagrama de secuencias:

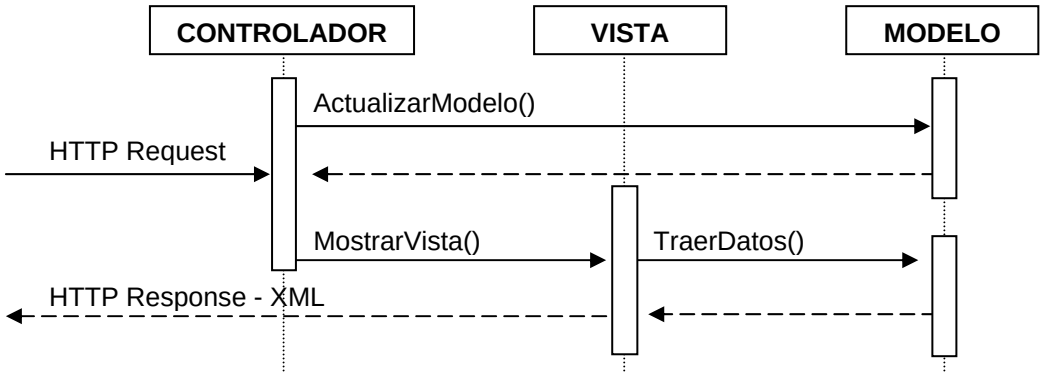


Figura III.2 - Diagrama de secuencia del patrón MVC del framework propuesto

III.3.3. Patrones de diseño en el MVC

La variante del patrón de arquitectura MVC planteado en el presente trabajo, incorpora al nivel de la capa Vista, el patrón de diseño *Abstract Factory* [5] (para gestionar la creación de las clases encargadas de generar los patrones de interface), y al nivel de la capa Modelo, el patrón *Observer* [5] (para notificar a ciertas funciones de los cambios producidos en el modelo).

III.3.3.1. Abstract Factory (Fábrica Abstracta)

Como ya se ha dicho, este patrón permite crear familias de objetos relacionados o dependientes entre sí, sin especificar sus clases concretas [5].

En el presente trabajo se considera sólo una familia de objetos que genera una única interface de usuario estandarizada. Ahora bien, supongamos que en el futuro surge la necesidad de incorporar uno o más estándares de interface con funcionalidades y diseño diferentes. El patrón Abstract Factory facilitaría la incorporación de esta nueva familia de objetos, por medio de una clase abstracta encargada de crear una interfaz para cada tipo básico de objeto de una familia. Luego una clase concreta para cada familia de objetos, implementa las operaciones encargadas de crear el objeto adecuado. En definitiva, un cliente crea objetos, a través de la interfaz de una clase concreta, sin conocer las clases que implementan dichos objetos para una familia de objetos específica (los clientes no se encuentran atados a una clase concreta, sino a la interfaz definida por una clase abstracta [5]).

A continuación se presenta la estructura en UML del patrón Abstract Factory

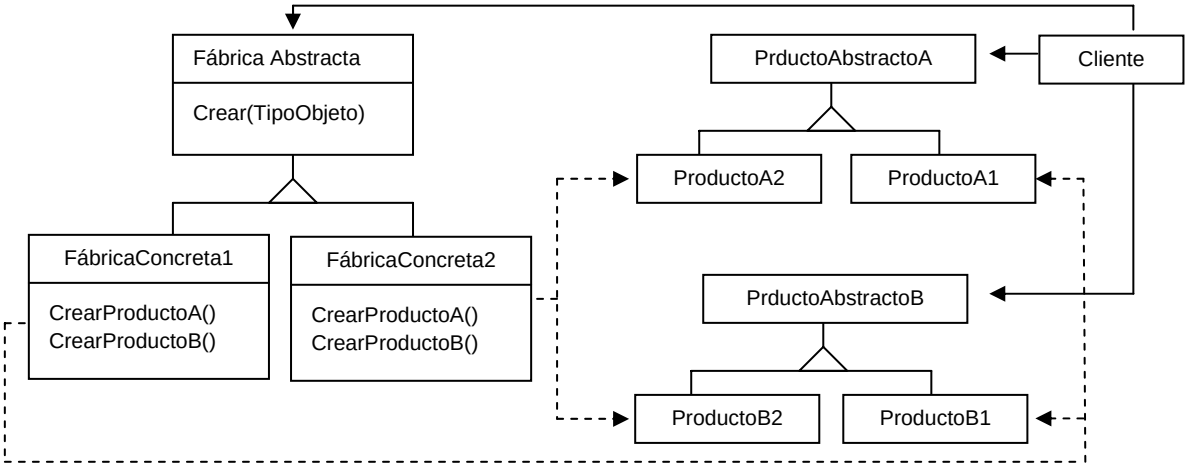


Figura III.3 - Estructura del patrón Abstract Factory [5]

Las siguientes figuras muestran el diagrama UML correspondiente a la variante planteada para el presente trabajo, Y el fragmento de código que lo implementa:

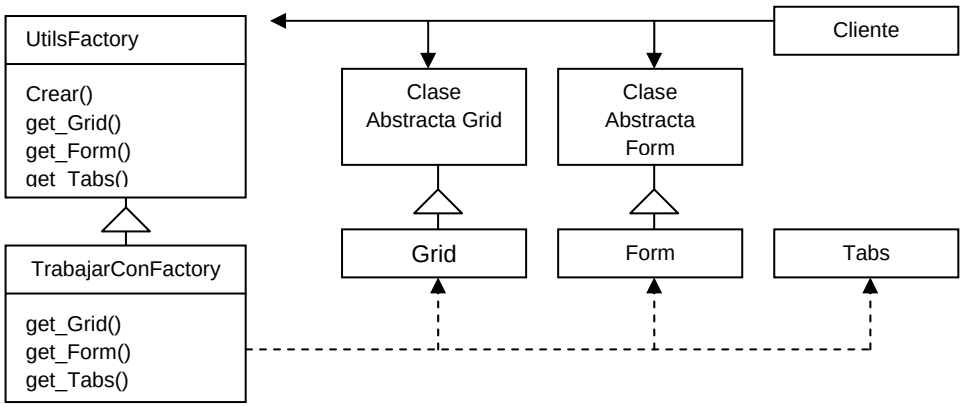


Figura III.4 – Variante del Patrón Abstract Factory para el Framework propuesto

Participantes:

UtilsFactory. Declara una interfaz para las operaciones que crean objetos en las clases abstractas Grid y Form

TrabajarconFactory. Implementa las operaciones para crear los objetos Grid, Form y Tabs

Clases Abstractas Grid y Form. Definen una interfaz para cada tipo de objeto (Grid y Form)

Grid y Form. Definen los objetos Grid y Form para que sean creados por TrabajarConFactory, e implementan las interfaces de las clases abstractas Grid y Form respectivamente.

Tabs. Define el objeto Tabs para que sea creado por TrabajarConFactory. No tiene asociada ninguna clase abstracta.

Cliente. Sólo usa las interfaces declaradas en UtilsFactory y en las clases abstractas Grid y Form

```
abstract class UtilsFactory{
        public $tipoObj;
        public function Crear(){
                return new $this->tipoObj();
        }
        abstract public function get_Grid();
        abstract public function get_Form();
        abstract public function get_Tabs();
}

class TrabajarConFactory extends UtilsFactory{
        public function get_Grid(){
                $this->tipoObj = "Grid";
                return $this->Crear();
        }
        public function get_Tabs(){
                $this->tipoObj = "Tabs";
                return $this->Crear();
        }
        public function get_Form(){
                $this->tipoObj = "Formulario";
                return $this->Crear();
        }
}
```

Figura III.5 – Fragmento de Código PHP para el patrón Abstract Factory

En la Figura III.5, *UtilsFactory* declara una interfaz para las operaciones que generan la familia de objetos que conforman los patrones de interacción del presente trabajo, mientras que *TrabajarConFactory* implementa las operaciones para la creación de dichos objetos.

III.3.3.2. Observer (Observador)

El patrón Observer define una dependencia de uno-a-muchos entre objetos, de manera que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él [5]. El patrón Observer permite definir un objeto observado, y uno o más objetos observadores que reaccionan a los cambios producidos en el primero.

A continuación se presenta la estructura en UML del patrón Observer

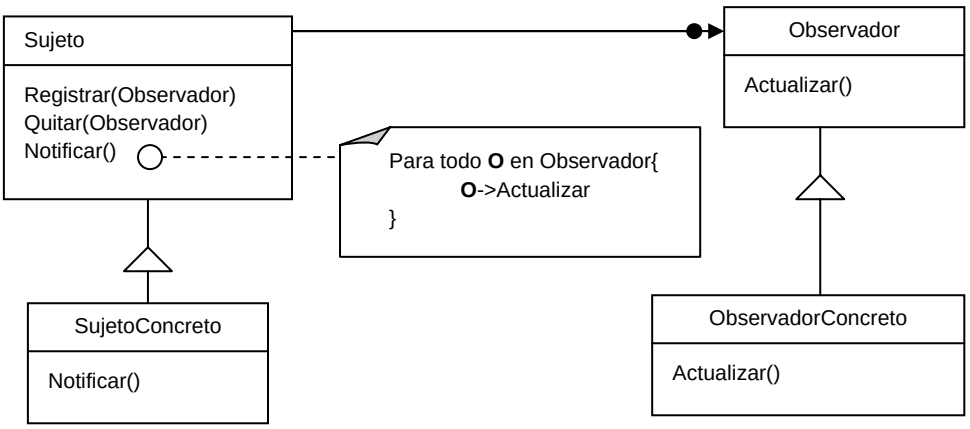


Figura III.6 - Estructura del patrón Observer [5]

Las siguientes figuras muestran el diagrama UML correspondiente a la variante planteada para el presente trabajo, Y el fragmento de código que lo implementa:

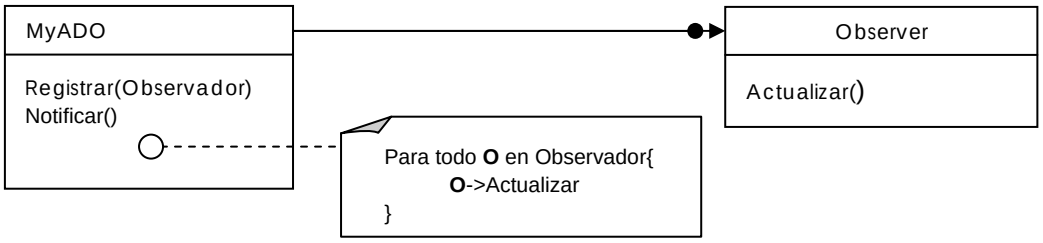


Figura III.7 – Variante del Patrón Observer para el Framework propuesto

Participantes:

MyADO. Proporciona un método para registrar N observadores, y otro para notificarlos de los cambios producidos en el modelo.

Observer. Implementa el método encargado de convocar la ejecución de cada uno de los N observadores registrados.

El sujeto observado es la clase MyADO (clase para la gestión de la bases de datos), en cuanto a los observadores, se tratan de funciones convocadas automáticamente al momento de producirse alguna actualización de datos. En todos los casos, MyADO sólo conoce una lista de referencia a los observadores, pero no conoce la implementación de los mismos, quedando esta a criterio de los programadores, y a las necesidades de negocio específicas.

```
class Observer{
        public static function Actualizar($Evento, $ID_Insert){
                $Evento($ID_Insert);
        }
}

function __construct(){
        $this->registrarEvento("BeforeInsert");
        $this->registrarEvento("AfterInsert");
}

private function RegistrarEvento($Evento){
        $this->Eventos[] = $Evento;
}

private function Notificar($Eventos, $Evento){
                foreach($this->Eventos as $Evento_A_Llamar){
                        if($Evento_A_Llamar == $Evento){
                                Observer::Actualizar($Evento, $this->ID_Insert);
                                break;
                        }
                }
}
```

Figura III.8 – Fragmento de Código para los métodos de registro y notificación de eventos

Supongamos, que se requiere mantener un log de actualización de una cierta tabla. Luego, siempre que se produzca un alta, la función *BeforeInsert* será convocada, quedando su implementación disponible para el código específico encargado de generar el log. Ver plantilla Lógica de Negocios de la sección III.6.3

III.3.4. Soporte AJAX

El presente framework dispone de un núcleo javascript que permite el diálogo asincrónico entre el cliente y el servidor mediante peticiones puntuales.

Al realizar una petición, el cliente queda a la espera de una respuesta. En el servidor, las peticiones se procesan, y generando las salidas en formato XML, las cuales son devueltas al cliente javascript quien en última instancia transforma las especificaciones XML en HTML produciendo la interface final para el usuario.

III.3.4.1. Consideraciones generales

La comunicación entre el cliente y el servidor es posible gracias al objeto **XMLHttpRequest()**. Este permite a través del método **“open”** enviar peticiones al servidor. Luego, mediante los métodos “**onreadystatechange”**, y **“readyState”**, se determina el estado de las peticiones. Por último, los métodos **“responseText”** y “**responseXML”**, obtienen la respuesta generada del lado del servidor.

```
var xmlHttp = new XMLHttpRequest(); //Referencia al Objeto XMLHttpRequest
if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)) {
   xmlHttp.open("GET", "TCProvincias.php", true);
   xmlHttp.onreadystatechange = function (){ //Se determina el estado de la petición
   if(xmlHttp.readyState == 0 || xmlHttp.readyState == 1 || xmlHttp.readyState == 2 || xmlHttp.readyState == 2) {
            document.getElementById(PlaceHolderID).innerHTML = "Sending Request...";
   }
   if (xmlHttp.readyState == 4){
            if (xmlHttp.status == 200){
                response = xmlHttp.responseText; // Lee la respuesta
                //Se ha producido algún error?
                if (response.indexOf("ERRNO") >= 0 || response.length == 0){
                        alert(response.length == 0 ? "Server serror." : response); // Muestra mensaje de error
                        return;
                }

                xmlResponse = xmlHttp.responseXML; // El servidor responde en formato XML
...
```

Figura III.9 - Fragmento de código javacript con la funcionalidad XMLHttpRequest

Una vez que el cliente obtiene la respuesta mediante **"responseXML"**, se aplica la transformación XSL mediante el objeto **XSLTProcessor()**, a través de los métodos **"importStylesheet"** y **"transformToFragment"**

```
xmlResponse = xmlHttp.responseXML; //El servidor responde en formato XML
 if (window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser){
            var xsltProcessor = new XSLTProcessor(); // Lee el documento XSLT
            xsltProcessor.importStylesheet("PatternTrabajarConXSL.xsl");

            //Genera el código HTML para la nueva página
            HTMLpage = xsltProcessor.transformToFragment(xmlResponse, document);
            var PlaceHolder = document.getElementById(PlaceHolderID); // Muestra la página
            PlaceHolder.innerHTML = "";
            PlaceHolder.appendChild(HTMLpage);
}
...
```

Figura III.10 - Fragmento de código javacript con la funcionalidad XSLTProcessor

En general, en el presente trabajo, los patrones de interacción se generan en el servidor, a partir de un conjunto de especificaciones XML, y posteriormente transformados a HTML por su correspondiente estilo XSL del lado del cliente.

A continuación, Las Figuras III.11 y III.12 muestran el fragmento de código XML que conforma la grilla en las pantallas *"Maestro de Registros"*, *"Selector Prompt"* y *"Vista General"*, y su correspondiente página de estilo.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<grid>
  <Keys>
     <key name="NotID" />
  </Keys>
  <Cols>
     <columna>ID</columna>
     <columna>FCH.PUBL.</columna>
     <columna>TITULO</columna>
     <columna>Editar</columna>
     <columna>Borrar</columna>
  </Cols>
  <NameForm>FNoticia</NameForm>
  <row>
     <elem name="NotID">1743</elem>
     <elem name="NotFchPub">2007-02-10</elem>
     <elem name="NotTitulo">La policía intervino para finalizar picadas de motos</elem>
  </row>
  <row>
     <elem name="NotID">1738</elem>
     <elem name="NotFchPub">2007-02-10</elem>
     <elem name="NotTitulo">Más de 20 mil personas vibraron con el Cosquín Rock</elem>
  </row>
</grid>
```

Figura III.11 - Especificación XML para una grilla del Framework propuesto

En el código anterior:

- La etiqueta <*grid*> indica el comienzo de la especificación XML para la grilla
- La etiqueta <*Keys*> indica el comienzo de el/los atributo(s) clave(s) de la grilla. Los atributos claves se indican mediante la etiqueta <*key name=”NomAttr” />*
- La etiqueta <*Cols*> hace referencia a las columnas de la grilla.
- La etiqueta <*NameForm*> hace referencia al formulario de edición asociado.
- Las etiquetas <*row*> hacen referencia a las filas de la grilla. Para cada etiqueta <*row*>, las etiquetas <*elem=”NomAttr”*> hacen referencia a un atributo asociado a una columna, y su correspondiente valor.

El fragmento de código XSL que transforma la grilla es:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<table>
<!-- 1) Cabecera de la grilla-->
<tr>
        <xsl:for-each select="grid/Cols/columna">
                <th><xsl:value-of select="." /></th>
        </xsl:for-each>
</tr>
<!-- 2) Filas de la grilla-->
<xsl:for-each select="grid/row">
        <xsl:choose>
                <xsl:when test="position() mod 2 = 0">
                        <tr class="trpar">
                                <xsl:call-template name="fila">
                                        <xsl:with-param name="linea" select="position()"/>
                                </xsl:call-template>
                        </tr>
                </xsl:when>
                <xsl:otherwise>
                        <tr>
                                <xsl:call-template name="fila">
                                        <xsl:with-param name="linea" select="position()"/>
                                </xsl:call-template>
                        </tr>
                </xsl:otherwise>
        </xsl:choose>
</xsl:for-each>
</table>
</xsl:stylesheet>
```

Figura III.12 – Página de Estilo XSL

Del código anterior se destacan dos secciones importantes:

1. **La construcción de la cabecera de la grilla:** Mediante una instrucción <*xsl:for-each*>, se recorren cada uno de los elementos del nodo <*Cols*>, y se imprimen los valores entre etiquetas HTML <th>

2. **Construcción de las filas de la grilla:** Mediante una instrucción <*xsl:for-each*>, se recorren cada uno de los elementos del nodo <*row*>. Con ayuda de la instrucción <*xsl:when test*> se calcula el numero de fila para pintar las filas pares de un color diferente a las filas impares. Por cada iteración dentro del ciclo principal <*xsl:for-each*>, una instrucción <*xsl:call-template*> convoca a la plantilla encargada de mostrar los valores para cada columna de la fila.

El resultado final es la grilla en formato HTML como se muestra en las figuras III.13 y III.14:

```
<table class="list">
<!-- Cabecera de la grilla-->
 <tr>
        <th>ID</th>
        <th>FCH.PUB</th>
        <th>TITULO</th>
        <th>Editar</th>
        <th>Borrar</th>
</tr>
<!-- Filas de la grilla-->
<tr class="trpar">
        <td>1743</td>
        <td>2007-02-10</td>
        <td>La policía intervino para finalizar picadas de motos</td>
        <td>Editar</td>
        <td>Borrar</td>
</tr>
<tr class="trpar">
        <td>1738</td>
        <td>2007-02-10</td>
        <td>Más de 20 mil personas vibraron con el Cosquín Rock</td>
        <td>Editar</td>
        <td>Borrar</td>
</tr>
</table>
```

Figura III.13 - Código resultante de la transformación XSL

ID	FCH.PUB	TITULO	Editar	Borrar
1743	2007-02-10	La policía intervino para finalizar picadas de motos	Editar	Borrar
1738	2007-02-10	Más de 20 mil personas vibraron con el Cosquín Rock	Editar	Borrar

Figura III.14 - Representación del código anterior en un Browser

III.3.5. Puntos calientes y puntos congelados

Toda la mecánica de navegación de las aplicaciones extendidas del framework viene dada por los patrones de interacción, implementados a partir de un conjunto de clases. A continuación, se definen:

a) para cada patrón de interacción, los puntos calientes encargados de configurar y especificar una aplicación en concreto; y

b) para cada clase, los puntos congelados que constituyen el núcleo del framework.

III.3.5.1. Puntos calientes (Hot Spots)

Patrón de Interacción	Clase	Puntos Calientes	Objetivo
Pantalla Maestro de Registros	Form	tabla (*)	Indica una tabla de una BD a partir de la cual generar un conjunto de filtros.
		AddAtributos	Define el conjunto de atributos para aplicar filtros sobre la grilla
		AddDefaultValues	Establece el valor por defecto para cada atributo definido mediante AddAtributos
		setTipoDBComboBox	Define un atributo como un Combo que obtiene datos a partir de una tabla de una BD.
		setTipoComboBox	Define un atributo como un Combo a partir de un conjunto de valores fijos
	Grid	AddGridKeys	Define el conjunto de atributos claves de la grilla
		FormAsoc (*)	Indica el nombre del Formulario asociado a la pantalla Maestro de Registros
		AddColumnasGrid	Define las columnas para la grilla
		AddColumnasSQL	Define el conjunto de atributos para generar la consulta SQL
		AddFiltro	Define el valor seleccionado para cada uno de los filtros indicados mediante AddFiltros
		getGridXML	Devuelve la grilla en formato XML
		getVistasXML	Establece una columna de la grilla como un punto de acceso a la pantalla “Vista general”

(*) Propiedades de las clases

Patrón de Interacción	Clase	Puntos Calientes	Objetivo
Pantalla Formulario	Form	tabla (*)	Indica una tabla de una BD a partir de la cual generar el formulario de actualización de registros.
		modo (*)	Indica el modo de acceso al formulario: Inserción, edición, eliminación.
		AddAtributos	Define el conjunto de atributos para generar el formulario de actualización
		AddAtributosClave	Define el conjunto de atributos clave
		setTipoDBComboBox	Define un atributo como un Combo que obtiene datos a partir de una tabla de una BD.
		setTipoComboBox	Define un atributo como un Combo a partir de un conjunto de valores
		AddDefaultValues	Define el valor por defecto para un atributo, cuando se accede al formulario en modo “Insertar”
		AddOnEditValues	Define el valor por defecto para un atributo, cuando se accede al formulario en modo “Edición”
		setPrompt	Establece para un atributo determinado un punto de acceso a la pantalla “Selector Prompt”

(*) Propiedades de las clases

Patrón de Interacción	Clase	Puntos Calientes	Objetivo
Pantalla VistaGeneral	Tabs	SelTab (*)	Indica la vista general actual
	Form	tabla (*)	Indica una tabla de una BD a partir de la cual generar el formulario con la vista general de atributos.

Patrón de Interacción	Clase	Puntos Calientes	Objetivo
Pantalla VistaGeneral (continuación)		AddTabs	Define el conjunto de Tabs, para acceder a cada una de las vistas
		AddKeysTabs	Define el conjunto de atributos clave de la “Vista General”
		AddAtributos	Define el conjunto de atributos para generar el formulario “Vista general”
		AddAtributosClave	Define el conjunto de atributos clave
		setTipoDBComboBox	Define un atributo como un Combo que obtiene datos a partir de una tabla de una BD.
		setTipoComboBox	Define un atributo como un Combo a partir de un conjunto de valores
	Grid	AddGridKeys	Define el conjunto de atributos claves de la grilla
		AddColumnasGrid	Define las columnas para la grilla
		AddColumnasSQL	Define el conjunto de atributos para generar la consulta SQL
		AddForeingKeys	Define las claves foráneas para cada una de las vistas
		getGridXML	Devuelve la grilla correspondiente a una vista, en formato XML

(*) Propiedades de las clases

Patrón de Interacción	Clase	Puntos Calientes	Objetivo
Pantalla Selector Prompt	Form	tabla (*)	Indica una tabla de una BD a partir de la cual generar un conjunto de filtros.
		AddAtributos	Define el conjunto de atributos para aplicar filtros sobre la grilla

Patrón de Interacción	Clase	Puntos Calientes	Objetivo
Pantalla Selector Prompt (continuación)		AddDefaultValues	Establece el valor por defecto para cada atributo definido mediante AddAtributos
		setTipoDBComboBox	Define un atributo como un Combo que obtiene datos a partir de una tabla de una BD.
		setTipoComboBox	Define un atributo como un Combo a partir de un conjunto de valores
	Grid	AddGridKeys	Define el conjunto de atributos claves de la grilla
		AddColumnasGrid	Define las columnas para la grilla
		FormAsoc (*)	Indica el nombre del Formulario asociado a la pantalla Maestro de Registros
		AddPromptReturn	Define el Atributo punto de retorno cuando un valor es seleccionado.
		AddColumnasSQL	Define el conjunto de atributos para generar la consulta SQL
		AddFiltro	Define el valor seleccionado para cada uno de los filtros indicados mediante AddFiltros
		getGridXML	Devuelve la grilla en formato XML
		getVistasXML	Establece una columna de la grilla como un punto de retorno de un valor para un atributo seteado como prompt

(*) Propiedades de las clases

III.3.5.2. Puntos Congelados (Frozen spots)

Clase	Puntos Conjelados	Objetivo
UtilsFactory (Clase Abstracta)	Crear	Implementa las operaciones para crear los diferentes objetos que conforman los patrones de interacción del framework
TrabajarConFactory	get_Grid	Se encarga de instanciar el objeto Grilla
	get_Filtro	Se encarga de instanciar el objeto Formulario
	get_Form	Se encarga de instanciar el objeto Formulario
	get_Tabs	Se encarga de instanciar el objeto Tabs
Form	readForm	Produce la salida final de un formulario en formato XML, en función de lo especificado por los diferentes puntos calientes
	getKeysXML	Retorna una especificación de un conjunto de claves principales en formato XML
	getFormXML	Retorna la especificación de un conjunto de atributos del formulario
Grid	getColumnasXML	Retorna la especificación de las diferentes columnas de la grilla en formato XML
	getKeysXML	Retorna una especificación de un conjunto de claves principales en formato XML
	getForeingKeysXML	Retorna una especificación de un conjunto de claves foráneas en formato XML
	setPromptReturn	Setea una columna dada como un punto de retorno de un valor (sólo para grillas dentro de una pantalla selector prompt)
	getParamsXML	Retorna en formato XML, la especificación del menú de navegación de una grilla.
Tabs	getKeysTabs	Retornan la especificación en formato XML, de los puntos de acceso a las vistas generales
	getTabsXML	

Clase	Puntos Conjelados	Objetivo
MyADO	getAttrTipos	Método encargado de realizar un mapeo entre una tabla de una Base de Datos, y el conjunto de atributos especificados mediante el método AddAtributos de la clase Form
	MySQLExec	Método encargado de ejecutar una consulta SQL
	MySQLExec	Método encargado de ejecutar una consulta SQL
	MakeFiltro	Métodos convocados por MySQLExec, encargados de configurar una consulta SQL
	countAllRecords	
	createSubpageQuery	
	getColumnasSQL	

III.4. Instanciación

Extender una aplicación a partir del framework propuesto, implica la generación de un conjunto de archivos con formato, sintaxis, nomenclatura y reglas bien definidas para poder poner en funcionamiento las diferentes clases que conforman la estructura fundamental del framework.

A continuación se define: a) la estructura de archivos que conforman el framework propiamente dicho, b) la sintaxis de cada uno de los puntos calientes definidos anteriormente, y finalmente c) los pasos necesarios para extender de forma correcta una aplicación en concreto.

III.4.1. Estructura de archivos

Extender una aplicación a partir del framework propuesto, implica la inclusión de la siguiente estructura de archivos al mismo nivel del proyecto a desarrollar:

Estructura (Carpeta / Archivos)	**Descripción**
Clases MyADO.php VistaGeneral.php dir.php Form.php Tabs.php Grid.php PatternFactory.php	Núcleo del framework encargado de: • gestionar los diferentes patrones de interacción • conexión con el motor de BD
FCKEditor	Directorio que contiene el conjunto de clases para la gestión transparente de código HTML como dato de entrada
Calendario	Directorio que contiene el conjunto de clases para la gestión de fechas como datos de entrada
Patrones PatternFormXSL.xsl PatternTrabajarConXSL.xsl PatternVistaGeneralXSL.xsl FormularioXSL.xsl PromptXSL.xsl GrillaXSL.xsl	Páginas de estilo encargadas de interpretar las especificaciones XML generadas por las vistas del MVC
Scripts	Directorio que contiene el núcleo AJAX, encargado de la gestión de las peticiones entre el cliente y el servidor
Index.php	Marco de trabajo o página principal de la aplicación extendida

III.4.2. Sintaxis para los Puntos Calientes

Clase	**Sintaxis**
Form	**$myForm->tabla** = “nombre_tabla” • *nombre_tabla:* Nombre de una tabla de la Base de datos

Clase	**Sintaxis**
Form (continuación)	**$myForm->AddAtributos**(“NomAttr”, TipoDato, Long, “Referencia”, 'Habilitado', 'ObjetoHTML', 'visible'); • *NomAttr:* Nombre de un atributo de la tabla indicada mediante “$myForm->tabla” • *TipoDato:* Tipo de dato del atributo (INTEGER, VARCHAR, etc.) • *Long:* Tamaño del campo en el formulario • *Referencia:* Nombre del atributo en el formulario • *Habilitado:* Indica si el atributo está habilitado o no, para edición (TRUE, FALSE) • *ObjetoHTML:* Objeto HTML del formulario (TEXTFIELD, SELECT, etc.) • *Visible:* Indica si el campo se encuentra visible o no (VISIBLE, HIDDEN)
	$myForm->AddDefaultValues('NomAttr', Valor**);** • *NomAttr:* Nombre de un atributo definido mediante AddAtributos() • *Valor:* Valor por defecto del atributo. Para el caso de un formulario de edición toma el valor solamente en modo INSERCIÓN. Para el caso de formularios de filtro se pasa el valor mediante $_GET[“NomAttr”]
	$myForm->SetTipoDBComboBox("NomAttrID", "AttrDesc", Valor, "tabla"**)**; • *NomAttrID:* Nombre del atributo identificador de la tabla “tabla” • *AttrDesc:* Nombre del atributo descripción de la tabla “tabla” • *Valor:* Valor del atributo pasado mediante $_GET[“AttrDesc”] • *tabla:* Nombre de la tabla a partir de la cual se genera la lista de valores
	$myForm->SetTipoComboBox("NomAttr", Lista, Valor**)** • *NomAttr:* Nombre del atributo • *Lista:* Lista de valores indicado como, “Val1:Desc1;Val2:Desc2;..;ValN:DescN” • *Valor:* Valor por defecto
	$myForm->AddAtributosClave("NomAttr", Valor**)**; • *NomAttr:* Atributo clave de la tabla indicada en “$myForm->tabla” • *Valor:* Valor del atributo pasado como $_GET[“NomAttr”]

Clase	Sintaxis
Form (continuación)	**$myForm->setPrompt(**"script_selector", "NomAttr"**)** • *script_selector:* Script del patrón "Pantalla Selector Prompt" • *NomAttr:* Establece en el atributo NomAttr un punto de acceso al "script_selector"

Clase	Sintaxis
Grid	**$myGrid->tabla** = "nombre_tabla" • *nombre_tabla:* Nombre de una tabla de la Base de datos
	$myGrid->FormAsoc = "script_formulario" • *script_formulario:* Script encargado de generar el formulario de edición asociado.
	$myGrid->AddGridKeys("NomAttr" **)** • *NomAttr:* Attributo clave de la tabla indicada mediante $myGrid->tabla
	$myGrid->AddColumnasGrid("NomAttr", "Referencia"**)** • *NomAttr:* Atributo columna de la grilla en una pantalla "Maestro de Registros" o "Prompt" • *Referencia:* Nombre de la columna
	$myGrid->AddColumnasSQL("NomAttr"**)** • *NomAttr:* Atributo que coincide en nombre y orden con los definidos mediante $myGrid->AddColumnasGrid("NomAttr", "Referencia")
	$myGrid->AddFiltro("NomAttr", Valor, TipoDato**)** • *NomAttr:* Atributo al que se desea aplicar filtro. Coincide con los atributos definidos en $myForm->AddAtributos(); • *Valor:* Valor del filtro pasado mediante $_GET["NomAttr"] • *TipoDato:* Tipo de dato del atributo (INTEGER, VARCHAR, etc.)

Clase	**Sintaxis**
Grid (continuación)	**$myGrid->getGridXML(**"", "", page**)** • Método que retorna la grilla especificada en formato XML. Dado un conjunto de registros divididos en páginas, el parámetro "page" definido mediante $_GET["page"], indica la página actual de dicho conjunto de registros
	$myGrid->getVistasXML("NomAttr", "srcipt_vistageneral"**)** • *NomAttr:* Nombre del atributo asociado a una columna de la grilla de una pantalla "Maestro de Registros" o "Vista General" • *srcipt_vistageneral:* Script que genera la vista general asociada
	$myGrid->AddPromptReturn("NomAttr", "NomAttrRet"**)** • *NomAttr:* Atributo de retorno para un valor asociado a NomAttr • *NomAttrRet:* Atributo que retorna un valor asociado a NomAttrRet Método válido únicamente para pantallas "Selector Prompt"

Clase	**Sintaxis**
Tabs	**$myTab->SelTab** = $Tab_Activo • *$Tab_Activo:* Variable con el valor de la vista actual
	$myTab->AddTabs("General:Tab1:Tab2:Tab_N"**);** • El argumento "General:Tab1:Tab2:Tab_N", hace referencia a los puntos de acceso a las diferentes vistas del registro General
	$myTab->AddKeysTabs("NomAttr", Valor**);** • *NomAttr:* Atributo clave para la vista General • *Valor:* Valor del atributo. El valor es pasado mediante $_GET["NomAttr"]

Clase	**Sintaxis**
MyADO	**$Ado->tablaBase** = "nombre_tabla" • *nombre_tabla:* Nombre de una tabla de la Base de datos

MyADO (continuación)	**$Ado->AddAtributosClave(**"NomAttr", Valor**);** • *NomAttr:* Atributo clave para la tabla “nombre_tabla” • *Valor:* Valor del atributo. El valor es pasado mediante $_GET[“NomAttr”]
	$Ado->AddAtributosInsert(**$_REQUEST, "NomAttr1;NomAttr2;...;NomAttr_N");** **$Ado->AddAtributosUpdate(**$_REQUEST, "NomAttr1;NomAttr2;...;NomAttr_N"**);** • *$_REQUEST:* Conjunto de valores pasados por un formulario • *NomAttr1;...;NomAttr_N:* Registros de la tabla “nombre_tabla”

III.4.3. Pasos para la generación de una aplicación concreta

A continuación se presentan los pasos básicos para extender una aplicación a partir del framework propuesto. Se asume que se dispone de un servidor Web (IIS, Apache, etc) que permita ejecutar aplicaciones basadas en PHP y MySQL

1. Inclusión de la estructura de archivos que conforman el núcleo del framework propuesto, al mismo nivel del proyecto a desarrollar.
2. Para cada tabla de la Base de Datos a mantener, crear los siguientes archivos, con la nomenclatura que se indica:
   - **Patrón Maestro de registros:** “TCon” + NombreTabla + “.php”
   - **Patrón Formulario:** “F” + NombreTabla + “.php”
   - **Lógica de negocio:** “P” + NombreTabla + “.php”
   - **Patrón Vista General:** “VGral” + NombreTabla + “.php”
   - **Patrón Selector Prompt:** “SEL” + NombreTabla + “.php”
3. Crear los archivos antes mencionados basados en las siguientes plantillas:

```
<?php
session_start();
$_SESSION['PageReturn'] = 'loadTCPage("'.$_SERVER['PHP_SELF'].'?'.$_SERVER['QUERY_STRING'].'",
"PlaceHolder")';
$_SESSION['PageReturnVG'] = '';

include("clases/PatternFactory.php");
include("clases/Grid.php");
include("clases/Form.php");
$page = $_GET["page"];
```

Figura III.15 - Plantilla Patrón Maestro de Registros

```
$ObjTC = new TrabajarConFactory();
$myForm = $ObjTC->get_Form();
$myForm->tabla = "nombre_tabla";
$myForm->modo = "INS";
//$myForm->AddAtributos(“NomAttr”, TipoDato, Long, “Referencia”, 'Habilitado', 'ObjetoHTML', 'visible');
//$myForm->AddAtributos(...);
//$myForm->AddDefaultValues(“NomAttr”, isset($_GET['NomAttr']) ? $_GET['NomAttr'] : "");
//$myForm->AddDefaultValues(...);

$myGrid = $ObjTC->get_Grid();
//$myGrid->tabla = "nombre_tabla";
//$myGrid->FormAsoc = "script_formulario";

//$myGrid->AddGridKeys('NomAttrID');
//$myGrid->AddGridKeys(...);
//$myGrid->AddColumnasGrid('NomAttrID', 'ID');
//$myGrid->AddColumnasGrid(...);
//$myGrid->AddColumnasSQL('NomAttr', 'NomAttr');//$myGrid->AddColumnasSQL(...);
//$myGrid->AddFiltro('NomAttr',  isset($_GET['NomAttr']) ? $_GET['NomAttr'] : "", TipoDato);
//$myGrid->AddFiltro(...);

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; //SALIDA ESPECIFICACIÓN XML
echo '<data>';
echo '<titulo>Titulo de Pantalla Maestro de Registros</titulo>';
echo '<Caller>TCPage</Caller>';
echo '<callerNamePag>script_TrabajarCon</callerNamePag>';
echo '<NameTC>script_TrabajarCon</NameTC>';
echo '<NameForm>script_formulario</NameForm>';
echo utf8_decode($myForm->getFormXML());
echo utf8_decode($myGrid->getGridXML("", "", $page));
echo $myGrid->getVistasXML('NomAttrID', ‘srcipt_vistageneral’);
echo '</data>';
?>
```

Figura III.15 - Plantilla Patrón Maestro de Registros (continuación)

```
<?php
include("clases/PatternFactory.php");
include("clases/Form.php");
include('FCKeditor/fckeditor.php');
$mod = isset($_GET['Mod']) ? $_GET['Mod'] : "";
switch($mod){
        case 'INS': $titulo = "Nueva Registro"; break;
        case 'UPD': $titulo = "Modificar Registro "; break;
        case 'DLT': $titulo = "Confirma borrar Registro ?";  break;
        default: $titulo = "Error. Comando desconocido...";
        exit;
}
$ObjTC = new TrabajarConFactory();//INSTANCIA CLASE FORM

$myForm = $ObjTC->get_Form();
$myForm->tabla = "nombre_tabla";
$myForm->modo = $mod;
//$myForm->AddAtributos(‘NomAttr1’, TipoDato, Long, 'Referencia', 'Habilitado', 'ObjetoHTML', 'visible');
//$myForm->AddAtributos(‘NomAttr_N’, TipoDato, Long, 'Referencia', 'Habilitado', 'ObjetoHTML', 'visible');
//$myForm->AddAtributosClave("NomAttr", isset($_GET["NomAttr"]) ? $_GET["NomAttr"] : "");
//$myForm->AddDefaultValues("NomAttr", Valor);
//$myForm->setPrompt("script_selector", "NomAttr");
$myForm->MakeFiltro();
$myForm->readForm();
```

Figura III.16 - Plantilla Patrón Formulario

```
echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; //SALIDA ESPECIFICACIÓN XML
echo '<data>';
echo '<titulo>'.$titulo.'</titulo>';
echo '<NameTC>script_TrabajarCon</NameTC>';
echo '<NameForm>script_formulario</NameForm>';
echo '<NameP>script_logica_negocio</NameP>';
echo '<modo>'.$mod.'</modo>';
echo utf8_decode($myForm->getFormXML($mod));
echo '</data>';
?>
```

Figura III.16 - Plantilla Patrón Formulario (continuación)

```
<?php
session_start();
include('clases/Form.php');
include ('clases/Observer.php');
include('clases/MyADO.php');

function BeforeInsert(){ //Código de usuario }
function BeforeUpdate(){//Código de usuario }
function BeforeDelete(){//Código de usuario }
function AfterInsert(){//Código de usuario }
function AfterUpdate(){//Código de usuario }
function AfterDelete(){//Código de usuario }
$mod = $_GET['Mod'];
if($mod!="cancel"){
        $Ado = new MyADO($mod);
        $Ado->tablaBase = " nombre_tabla ";
        $Ado->getAttrTipos();
        //$Ado->AddAtributosClave("NomAttr", isset($_GET["NomAttr"]) ? $_GET["NomAttr"] : "");
        switch($mod){
                case 'INS': $Ado->AddAtributosInsert($_REQUEST, "NomAttr1;...;NomAttr_N"); break;
                case 'UPD': $Ado->AddAtributosUpdate($_REQUEST, "NomAttr1;...;NomAttr_N"); break;
                case 'DLT': break;
                default:echo 'Server error: client command missing.'; exit;
        }
        if ($Ado->MySQLExec()){ echo $_SESSION["PageReturn"];}else echo "Err";
}else echo $_SESSION["PageReturn"];
?>
```

Figura III.17 - Plantilla Lógica de Negocios

```
<?php
session_start();
$_SESSION['PageReturnVG'] =
'loadVistaGral("'.$_SERVER['PHP_SELF'].'?'.$_SERVER['QUERY_STRING'].'", "PlaceHolder")';

include("clases/PatternFactory.php");
include("clases/Grid.php");
include("clases/Formphp");
include("clases/Tabs.php");

$tab = isset($_GET['Tab']) ? $_GET['Tab'] : "";
$pag = isset($_GET['page']) ? $_GET['page'] : "1";
```

Figura III.18 - Plantilla Patrón Vista General

```
$ObjTC = new TrabajarConFactory();
$myTab = $ObjTC->get_Tabs();
$myTab->SelTab = $tab;
$myTab->AddTabs("General:Tab1:...:Tab_N");
$myTab->AddKeysTabs("NomAttr", isset($_GET["NomAttr"]) ? $_GET["NomAttr"] : "");

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; //SALIDA ESPECIFICACIÓN XML
echo '<data>';
echo '<Caller>script_TrabajarCon</Caller>';
echo '<callerNamePag>script_VistaGeneral</callerNamePag>';
echo '<NameP>script_LogicaNegocio</NameP>';
echo '<NameForm>script_Formulario</NameForm>';
echo '<TabActivo>'.$myTab->SelTab.'</TabActivo>';
echo '<VtaGral>'.$myTab->getTabsXML().'</VtaGral>';

switch($tab){
    case "General": //INSTANCIA CLASE FORM
            $myForm = $ObjTC->get_Form();
            //$myForm->tabla = "nombre_tabla";
            //$myForm->AddAtributos("...", TipoDato, Long, “Referencia”, 'Habilitado', 'ObjetoHTML', 'visible');
            //$myForm->AddAtributosClave("NomAttr", isset($_GET["NomAttr"]) ? $_GET["NomAttr"] : "");
            $myForm->MakeFiltro();
            $myForm->readForm();
            echo $myForm->getFormXML('DSP');
    break;
    case "Tab1":
            $myGrid = $ObjTC->get_Grid();
            $myGrid->tabla = "nombre_tabla";
            $myGrid->AddGridKeys('NomAttr');
            //$myGrid->FormAsoc = "script_formulario";
            //$myGrid->AddGridKeys(...);
            //$myGrid->AddColumnasGrid(...);
            //$myGrid->AddColumnasSQL(...);
            //$myGrid->AddForeingKeys('NomAttr', isset($_GET['NomAttr']) ? $_GET['NomAttr'] : "", TipoDato);
            echo $myGrid->getGridXML("", "", $pag);
     break;
     ...

     case "Tab_N":
            $myGrid = $ObjTC->get_Grid();
            $myGrid->tabla = "nombre_tabla";
            $myGrid->AddGridKeys('NomAttr');
            //$myGrid->FormAsoc = "script_formulario";
            //$myGrid->AddGridKeys(...);
            //$myGrid->AddColumnasGrid(...);
            //$myGrid->AddColumnasSQL(...);
            //$myGrid->AddForeingKeys('NomAttr', isset($_GET['NomAttr']) ? $_GET['NomAttr'] : "", TipoDato);
            echo $myGrid->getGridXML("", "", $pag);
     break;
}
echo '</data>';
?>
```

Figura III.18 - Plantilla Patrón Vista General (continuación)

```
<?php
include("clases/PatternFactory.php");
include("clases/Grid.php");
include("clases/Form.php");

$page = $_GET["page"];
```

Figura III.19 - Plantilla Patrón Selector Prompt

```
$ObjTC = new TrabajarConFactory();

$myForm = $ObjTC->get_Form();
$myForm->tabla = "nombre_tabla";
$myForm->modo = "INS";
//$myForm->AddAtributos("...", TipoDato, Long, "Referencia", 'Habilitado', 'ObjetoHTML', 'visible');
//$myForm->AddDefaultValues(NomAttr, isset($_GET['NomAttr']) ? $_GET['NomAttr'] : "");

$myForm->readForm();

$myGrid = $ObjTC->get_Grid();
//$myGrid->tabla = "nombre_tabla";
//$myGrid->FormAsoc = "script_Formulario";
//$myGrid->AddGridKeys(...);

//$myGrid->AddColumnasGrid(...);
//$myGrid->AddPromptReturn('NomAttr','NomAttrRetorno');
//$myGrid->AddColumnasSQL(...);
//$myGrid->AddFiltro(...);

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; //SALIDA ESPECIFICACIÓN XML
echo '<data>';
echo '<titulo>Titulo de Pantalla Maestro de Registros</titulo>';
echo '<Caller>PromptPage</Caller>';
echo '<callerNamePag>script_Selector</callerNamePag>';
echo '<NameTC>script_Selector</NameTC>';
echo utf8_decode($myForm->getFormXML());
echo utf8_decode($myGrid->setPromptReturn());
echo utf8_decode($myGrid->getGridXML("", "", $page));
echo '</data>';
?>
```

Figura III.19 - Plantilla Patrón Selector Prompt (continuación)

Para completar las plantillas, se deberá seguir la sintaxis de los puntos calientes como se indica en el apartado III.6.2.

En el **ANEXO A**, se describen los pasos para la utilización del framework aplicado a un caso práctico.

III.5. Aplicación del Framework

El Framework obtenido, se aplicó con éxito en el desarrollo de un conjunto de aplicaciones Web de gestión de contenidos, destinado a una empresa de multimedios de nuestra provincia.

Los requisitos funcionales de cada aplicación, fueron aportados por un grupo de periodistas que se desempeña desde hace un tiempo en el ámbito de la publicación online de diarios digitales:

- Generar aplicaciones fáciles de usar con interfaces sencillas y ligeras
- Generar código HTML de manera transparente

- Proporcionar mecanismos de búsqueda y filtros, para la recuperación rápida de la información
- Proporcionar mecanismos que permitan relacionar de manera sencilla los diferentes contenidos multimedia (audio, video, imágenes)
- Proporcionar mecanismos para la publicación automática de Encuestas online
- Prever mecanismos de auditoria en el uso del sistema
- Proporcionar roles de usuario para el sistema

Capítulo 4

Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web

Evaluación de Resultados

IV.1. Introducción

En este capítulo, se tomarán como referencia los valores de tiempo y costo reales obtenidos en el proceso de desarrollo de la aplicación de gestión de contenidos a partir del framework, y se los contrastará contra los valores estimados obtenidos de aplicar los modelos COCOMO y COCOMO II. La idea, es poder comparar los tiempos de desarrollo de una aplicación, como una extensión del framewrok, contra lo que se podría llamar un desarrollo tradicional.

Además, se evaluarán características que hacen a la calidad de la aplicación extendida, y el grado de dificultad en el uso del framework durante el proceso de desarrollo.

IV.2. Aplicación Extendida Vs. Desarrollo Tradicional

El software de gestión de contenidos Web extendida a partir del framework, con los requisitos funcionales indicados en la sección III.7, se llevó a cabo por un equipo de desarrollo integrado por una sola persona. Dispone en su versión final de unas 13505 líneas de código (incluido el núcleo del framework) y unas 6959 líneas la aplicación extendida únicamente; y la duración total del proyecto hasta su puesta en marcha demandó unos 45 días aproximadamente.

Para contrastar los valores reales obtenidos respecto a los que podrían obtenerse mediante un desarrollo tradicional, se realizará una estimación utilizando COCOMO, tomando como dato de entrada el total de líneas de código del proyecto.

Para este caso, se opta por el modelo básico y el modo orgánico, propio de un proyecto pequeño, en donde las ecuaciones se definen de la siguiente manera:

$E = ab (KLDC)^{bb}$

$D = cb (E)^{db}$

$P = E / D$

Dónde **E** es el esfuerzo aplicado en persona-mes, **D** el tiempo de desarrollo en meses, **KLDC** el número de líneas (en miles) estimadas para el proyecto, y **P** el número de personas necesarias.

Según la Figura II.9 de la sección II.3.2, los coeficientes **ab**, **bb**, **cb** y **db** para el modo orgánico son los siguientes:

ab = 2,4; bb = 1,05; cb = 2,5; db = 0,38

Luego, los valores estimados tomando como entrada el número total de líneas incluidas las líneas del framework son:

E = 2,4 (13,505) ^{1,05} = **36,91 mes/persona**

D = 2,5 (36,91) ^{0,38} = **9,8 meses**

P = 36,91 / 9,8 = 3,7 ≡ **4 personas**

Considerando únicamente el total de líneas de la aplicación extendida, los valores estimados resultan:

E = 2,4 (6,959) ^{1,05} = **18,40 mes/persona**

D = 2,5 (18,40) ^{0,38} = **7,5 meses**

P = 18,40 / 7,5 = 2,4 ≡ **2 personas**

A continuación, se aplicará el modelo COCOMO II tomando como puntos de objeto, las pantallas o patrones de interacción extendidas del framework:

	Patrones de interacción				
	TrabajarCon	**Formulario**	**Vista Gral.**	**Selector Prompt**	**Lógica de Negocio**
Total	13	20	5	12	17
Complejidad	1	1	2	1	1

El total de puntos de objeto se determina con la sumatoria de multiplicar el número original de pantallas, por el peso del factor de complejidad asignado.

Total PO = ∑ (nro. instancias x peso)

Esto es:

Total PO = (13x1) + (20x1) + (5x2) + (12x1) +(17x1)

Total PO = 13 + 20 + 5 + 12 + 17 = 67

El esfuerzo estimado, expresado en Meses/Persona se define como:

E = PO / PROD; Dónde PROD (de acuerdo a la Figura II.12 de la sección II.3.2.1), se define como:

PROD = (ECD + MCE) = 25 + 13 = 38

Considerando para este caso, ECD (la experiencia/capacidad del desarrollador) alta, y MCE (Madurez/capacidad del entorno) normal. Luego,

E = 67 / 38 = 1,7 meses/persona

Por último, y con el valor estimado E, es posible obtener aplicando COCOMO tradicional, los valores para:

D = cb (E) $^{\mathbf{db}}$ = 2,5 (1,7) ^{0,38} = **3,05 meses**

P = E / D = 1,7 / 3,05 = 0,55 ≡ **1 persona**

En éste caso, la estimación se realizó sobre la base de valores conocidos como la cantidad total de pantallas extendidas del framework, sin embargo este dato, puede ser fácilmente calculado a priori, en función del número de tablas de la Base de Datos. En efecto, el framework propuesto, presupone la extensión de 5 archivos o plantillas por cada tabla a mantener, lo cual no siempre el cierto ya que las pantallas *"Vista General"* se utilizan en aquellos casos en que es necesario modelar una relación del tipo 1:N; las pantallas *"Selector Prompt"* en los casos en que se requieran búsquedas sobre un conjunto grande de registros, y las pantallas *"Maestro de Registros"* resultan útiles como acceso principal a las tablas maestras.

No obstante, conociendo el número total de tablas del modelo de datos, el total de tablas con al menos una relación 1:N, y el número de tablas maestras, es posible estimar la variable Total PO, como sigue: **Total PO = CVG + RESTO**.

Dónde,

CVG = (Nro. Tablas con relación 1:N) x (Peso complejidad media). Representa el número total de pantallas Vista General multiplicado por 2

RESTO = [(Nro. Maestros)+(Nro. Tablas x 3)] x (Peso complejidad simple). Representa el número total de pantallas distintas a las pantallas Vista General.

Luego, si el modelo de datos consta de 23 tablas, de las cuales 9 son tablas maestro, y 5 tienen alguna relación tipo 1:N; entonces:

Total PO = (5 x 2) + [9 + (23 x 3)] x 1

Total PO = 10 + [9 + 69] = 88

Total PO = 10 + 78 = 88

Con el valor de Total PO calculado, es posible estimar el esfuerzo como sigue:

PROD = (ECD + MCE) = 25 + 13 = 38

E = 88 / 38 = 2,3 meses/persona

D = cb (E) $^{\mathbf{db}}$ = 2,5 (2,3) ^{0,38} = **3,43 meses**

P = E / D = 2,3 / 3,43 = 0,67 ≡ **1 persona**

A continuación se presenta un cuadro comparativo con los valores calculados, y los valores reales obtenidos.

	Duración	Personas
Valor Real	1,5 meses	1
COCOMO II	3,05 meses	1
COCOMO II (a priori)	3,43 meses	1
COCOMO Tradicional	7,5 meses	2

Se observa que los valores obtenidos mediante el modelo COCOMO II resultan más próximos a los valores reales.

El costo de todo proyecto, se encuentra siempre ligado a factores de esfuerzo, duración y número de personas que intervienen en el proceso de desarrollo. El mismo pesenta también, un grado de subjetividad y está sujeto a la capacidad de los desarrolladores y a las diferentes situaciones propias del entorno donde se desempeñan.

IV.3. Factores de Calidad del Framework y sus aplicaciones extendidas

Pressman [20] define a la calidad como la concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y los requerimientos implícitos que desea el usuario y que no fueron establecidos formalmente.

Según McCall existen un gran número de factores que afectan a la calidad de un producto software. Están aquellos que pueden medirse directamente, como los defectos por punto de función, y los que pueden medirse indirectamente, como la usabilidad y facilidad de mantenimiento. McCall propone tres grupos para clasificar los factores que afectan a la calidad de un producto software, y tienen que ver con sus características operativas, su capacidad de cambios, y su adaptabilidad a nuevos entornos. (Figura IV.1)

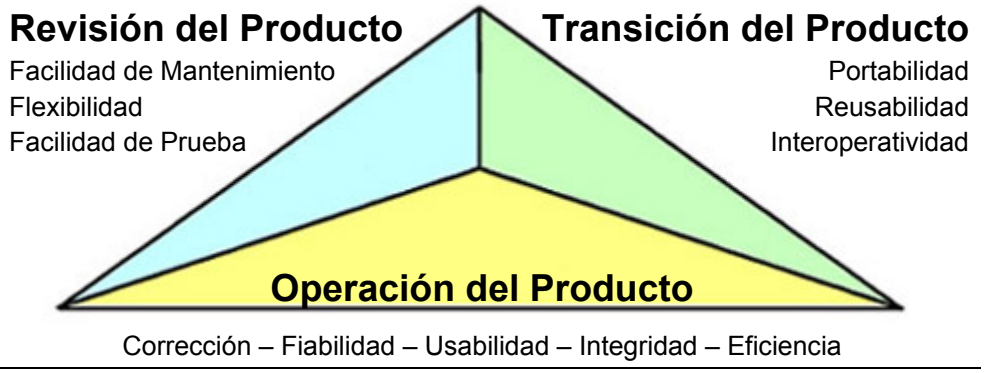


Figura IV.1 - Factores de calidad de un producto software [20]

En el presente trabajo, se evalúan:

- La facilidad en el uso y aplicación del framework; y
- La facilidad de uso y mantenimiento de las aplicaciones extendidas

IV.3.1 Facilidad en el uso del Framework

Si bien, el estado de desarrollo del framework propuesto se encuentra en una etapa inicial, dónde no existen procesos para la generación automática de código y validación del mismo, ni tampoco se cuenta con una interface gráfica de usuario que asista en el proceso de desarrollo; el mismo se puso a consideración y uso de tres programadores con niveles intermedios en el desarrollo de aplicaciones Web.

Cada uno de los programadores fue provisto de un ambiente de desarrollo con los elementos necesarios para llevar a cabo el experimento:

- Servidor Web con PHP, y MySQL instalados y configurados
- Una base de datos conteniendo dos tablas, provincias y ciudades, con una relación 1:N entre la primera y la segunda

- Una breve introducción sobre el uso del framework, junto a toda la documentación indicada en las secciones III.3 y III.4

El experimento consistió en:

1. Solicitar a los programadores generar una aplicación que permita mantener y recuperar datos de las tablas provincias y ciudades, de acuerdo a los siguientes requisitos funcionales (ver Anexo A):

   - Alta, baja y modificación de registros
   - Búsquedas de provincias y ciudades
   - Filtro de ciudades por una provincia específica

   Para completar este punto, los programadores dispusieron de 5hs reloj.

2. Una vez finalizado el primer punto, se los solicito completar una encuesta como la mostrada en la Figura IV.2.

Encuesta Dificultad en el uso del framework

1) ¿ Pudo completar los requerimientos funcionales del punto 1 ?

☐ a) No ☐ b) A medias ☐ c) Si. Sin problemas

2) ¿ Como encuentra el proceso de extensión del framework ?

☐ a) Complejo ☐ b) Fácil de aplicar

3) ¿ Cómo calificaría la aplicación generada ?

☐ a) Compleja ☐ b) Intuitiva y fácil de utilizar

4) ¿ Considera que la herramienta podría ayudar a minimizar los tiempos en el proceso de desarrollo ? Indique el porque de su respuesta.

☐ a) No ☐ b) Si ☐ c) Si. Pero sólo en proyectos pequeños

5) ¿ Utilizaría la herramienta para proyectos propios ? Indique el porque de su respuesta.

☐ a) No ☐ b) Si ☐ c) Si. Pero sólo en proyectos pequeños

☐ d) Lo pondría a consideración

OSERVACIONES/SUGERENCIAS:

_____

Figura IV.2 - Encuesta dificultad en el uso del framework

A cada alternativa de respuesta le fue asignado un peso: *positivo*, *negativo* y *neutro*, como se aprecia en la Figura IV.3

ITEM	RESPUESTA	VALOR
1	**a)**	Negativo
	b)	Neutro
	c)	Positivo
2	**a)**	Negativo
	b)	Positivo
3	**a)**	Negativo
	b)	Positivo
4	**a)**	Negativo
	b)	Positivo
	c)	Positivo
5	**a)**	Negativo
	b)	Positivo
	c)	Positivo
	d)	Neutro

Figura IV.3 - Tabla de pesos

Los resultados obtenidos se muestran en las siguientes tablas:

	Ítem	**Respuesta**	**Valor**
Programador 1	1	c)	Positivo
	2	b)	Positivo
	3	b)	Positivo
	4	c)	Positivo
	5	d)	Neutro
Totales	Positivos: 4; Negativos: 0; Neutros: 1		

	Ítem	**Respuesta**	**Valor**
Programador 2	1	b)	Neutro
	2	a)	Negativo
	3	b)	Positivo
	4	a)	Negativo
	5	a)	Negativo
Totales	Positivos: 1; Negativos: 3; Neutros: 1		

	Ítem	**Respuesta**	**Valor**
Programador 3	1	c)	Positivo
	2	b)	Positivo
	3	b)	Positivo
	4	b)	Positivo
	5	c)	Positivo
Totales	Positivos: 5; Negativos: 0; Neutros: 0		

Sobre un universo encuestado de tres desarrolladores, y un total de 15 respuestas obtenidas de las cuales 10 resultaron positivas (66,66%), 3 negativas (20%) y 2 neutras (13,33%); contar con casi un 70% de respuestas favorables, marca un grado bastante alentador de aceptación y facilidad de uso del framework en cuestión.

En cuanto a las observaciones y/o sugerencias, estuvieron orientadas hacia la posibilidad de contar con:

- Procesos para automatizar la generación de código y prevenir errores
- Un entorno visual que asista al programador en la tarea de desarrollo (mapeo automático de tablas, selección visual de campos a validar, incorporación de más objetos HTML como checkbox, radiobutton, etc.)

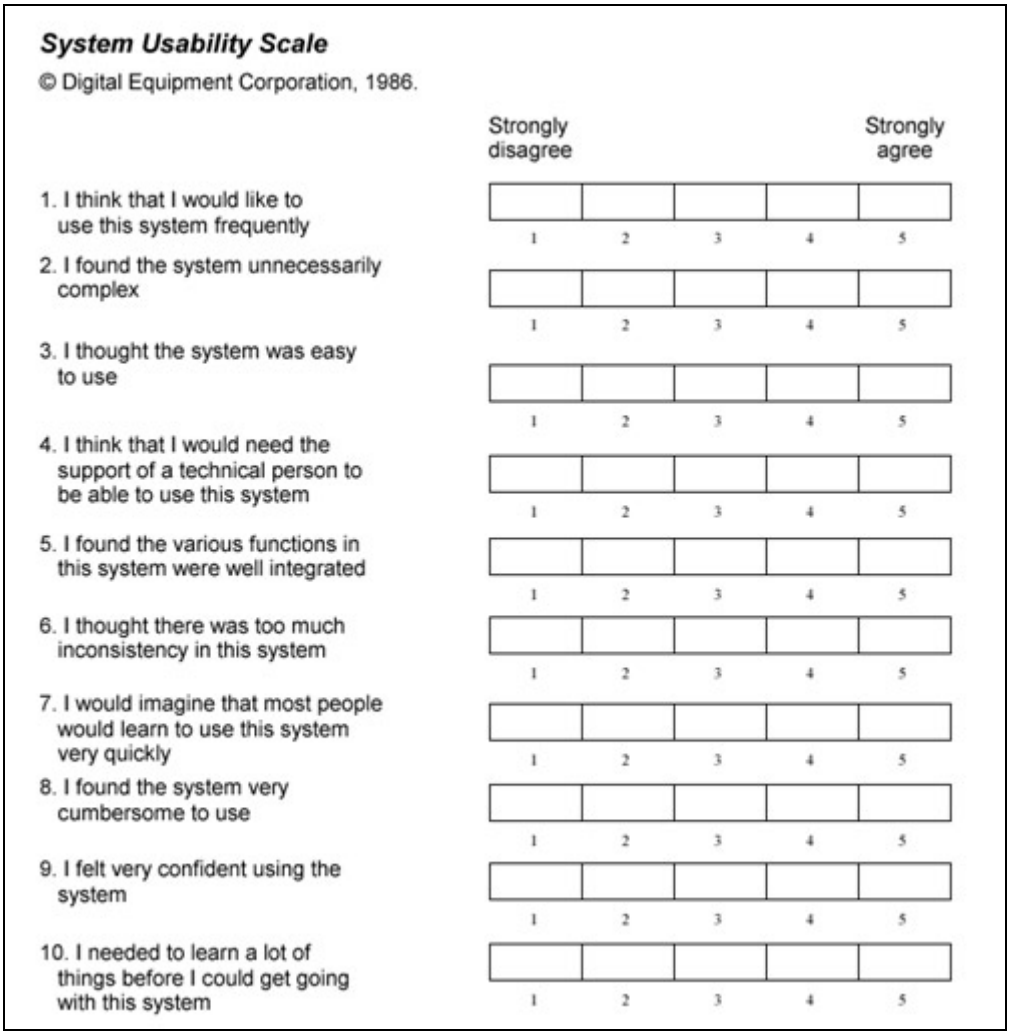
Estos puntos, son características que influyen directamente sobre la usabilidad de cualquier herramienta de desarrollo incluido el framework en cuestión.

IV.3.2 Usabilidad las aplicaciones extendidas

La aceptación final de una aplicación software por parte de un usuario, depende en gran medida de la percepción que éste tenga del sistema y esta percepción se logra mediante la interfaz del sistema. Por lo tanto diseñar interfaces fáciles de aprender y de usar, robustas y flexibles es un requisito fundamental para el éxito de un producto software.

Existen diferentes técnicas para medir el grado de usabilidad de un sistema, y en general se basan en la estimación de características tales como, el tiempo que le lleva un usuario comenzar a hacer un uso eficiente del sistema, aumento de la productividad, pasos para llevar a cabo una tarea específica, valoración subjetiva mediante el uso de cuestionarios.

En éste caso se hará uso de *SUS* (System Usability Scale) [1] Figura IV.4. *SUS* proporciona un cuestionario fácil de completar y evaluar, y se utiliza una vez que los usuarios han comenzado a utilizar el sistema. *SUS* arroja valores que van de 0 a 100, proporcionando una medida subjetiva compuesta, de la usabilidad de todo el sistema.



System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Figura IV.4 – SUS. System Usability Scale [1]

Para calcular el puntaje final del cuestionario,

1) Se deben sumar los puntajes asignados a cada ítem, los que varían en un rango de 0 a 4, y en donde además

   a) para los ítems 1, 3, 5, 7 y 9, se debe restar 1 al valor del rango elegido.

   b) para los ítems 2, 4, 6, 8, y 10, se debe restar 5 entre el valor del rango elegido.

2) Finalmente se multiplica la sumatoria antes obtenida por el valor 2,5

El cuestionario se realizó sobre una muestra de siete usuarios del sistema Web de gestión de contenidos extendido del framework. A continuación se presenta el cuestionario realizado por uno de los usuarios encuestados.

	Completamente en desacuerdo				Completamente de acuerdo
	1	2	3	4	5
1. Me gustará usar este sistema frecuentemente				√	
Puntuación: 4 – 1 = **3**					
2. Encuentro al sistema innecesariamente complejo	√				
Puntuación: 5 – 1 = **4**					
3. Pienso que el sistema es fácil de usar					√
Puntuación: 5 – 1 = **4**					
4. Creo que necesitaría del soporte técnico para hacer uso del sistema	√				
Puntuación: 5 – 1 = **4**					
5. Creo que las diferentes funciones de este sistema están bien integradas				√	
Puntuación: 4 – 1 = **3**					
6. Creo que existen demasiadas inconsistencias en el sistema	√				
Puntuación: 5 – 1 = **4**					
7. Creo que la mayoría de las personas pueden aprender a usar rápido el sistema				√	
Puntuación: 4 – 1 = **3**					
8. Me pareció que el sistema es muy lento en su uso	√				
Puntuación: 5 – 1 = **4**					
9. Me siento seguro usando el sistema				√	
Puntuación: 4 – 1 = **3**					
10. Necesito aprender muchas cosas antes de poder usar este sistema		√			
Puntuación: 5 – 2 = **3**					

El puntaje obtenido del cuestionario es de 35 puntos, Luego: *SUS* = 35 x 2,5 = 87,5

Los puntajes totales reales fueron:

Usuario 1 = 87,5 puntos

Usuario 2 = 100 puntos

Usuario 3 = 72,5 puntos

Usuario 4 = 87,5 puntos

Usuario 5 = 87,5 puntos

Usuario 6 = 87,5 puntos

Usuario 7 = 100 puntos

Los puntajes finales aportados, permitieron obtener un valor promedio para *SUS* de 88,92; lo cual nos indica un alto grado usabilidad y aceptación del sistema por parte de los usuarios del mismo.

IV.3.3 Facilidad de mantenimiento de las aplicaciones extendidas

Se puede definir a la facilidad de mantenimiento de un producto software, como la facilidad con la que un programa puede ser corregido para adaptar el sistema a los cambios del entorno, o a los nuevos requerimientos de usuario del sistema.

Existen diferentes métricas que permiten estimar la facilidad de mantenimiento. Una métrica simple, es la métrica orientada al tiempo [20], que consiste en promediar las mediciones de tiempo que se tarda en el análisis del cambio, diseño e implementación del cambio, y la prueba y puesta en producción del mismo. Luego, a partir de esta medición es posible concluir que los programas con un tiempo medio de cambio bajo resultan fáciles de mantener.

Desde la puesta en producción del Sistema Web de Gestión de contenidos, el mismo ha venido sufriendo numerosos cambios, la mayoría de ellos relacionados directamente a modificaciones en la estructura de datos del sistema (incorporación de nuevas tablas y atributos, cambio en tipos de datos, etc.)

Gracias a la estructura y el soporte proporcionados por el framework (patrones de diseño, de arquitectura e interacción, plantillas y sintaxis bien definidas para la creación de interfaces), la mayoría de los cambios mencionados pudieron aplicarse en tiempos muy breves, con lo cual el tiempo medio de cambio resultó relativamente bajo.

Por ejemplo (en referencia al caso práctico del ANEXO A). Si se considera la necesidad de incorporar a la tabla ciudades, el atributo *CodigoPostal*, bastaría con modificar los archivos FCiudades.php y PCiudades.php como sigue:

```
<?php
...
...

$ObjForm = new TrabajarConFactory();
$myForm = $ObjForm->get_Form();
$myForm->tabla = "ciudades";
$myForm->modo = $mod;
$myForm->AddAtributos('CiudadID', INTEGER, 5, 'CIUDAD.ID', 'FALSE','TEXTFIELD', 'visible');
$myForm->AddAtributos('Ciudad', VARCHAR, 70, 'CIUDAD', 'TRUE','TEXTFIELD', 'visible');
$myForm->AddAtributos('CodigoPostal', INTEGER, 8, 'COD.POSTAL', 'TRUE','TEXTFIELD', 'visible');
$myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROV.ID', 'TRUE','DBComboBox', 'visible');
...
...
?>
```

Figura IV.5 - FCiudades.php

```
<?php
...
...
switch($mod){
        case 'INS': $Ado->AddAtributosInsert($_REQUEST, "CiudadID;Ciudad;CodigoPostal;ProvinciaID");
        break;
        case 'UPD': $Ado->AddAtributosUpdate($_REQUEST, " CiudadID;Ciudad;CodigoPostal;ProvinciaID ");
        break;
        case 'DLT': break;
        default:echo 'Server error: client command missing.';
        exit;
}
...
...
?>
```

Figura IV.6 - PCiudades.php

Para el caso que se requiera incorporar búsquedas de ciudades por medio del campo CodigoPostal, se debería modificar el archivo TConCiudades.php como se muestra en el código de la siguiente figura:

```
<?php
...
...
$myForm->AddAtributos('CiudadID', INTEGER, 5, 'CIUDAD.ID.', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddAtributos('Ciudad', VARCHAR, 30, 'CIUDAD', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddAtributos('CodigoPostal', INTEGER, 8, 'CODIGO POSTAL', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROVINCIA', 'TRUE', 'DBComboBox', 'visible');

$myForm->AddDefaultValues('CiudadID',  isset($_GET['CiudadID']) ? $_GET['CiudadID'] : "");
$myForm->AddDefaultValues('Ciudad',  isset($_GET['Ciudad']) ? $_GET['Ciudad'] : "");
$myForm->AddDefaultValues('CodigoPostal',  isset($_GET['CodigoPostal']) ? $_GET['CodigoPostal'] : "");
$myForm->AddDefaultValues('ProvinciaID',  isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "");

$myForm->setTipoDBComboBox('ProvinciaID',  'Provincia', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "",
'provincias');
$myForm->readForm();

$myGrid = $ObjTC->get_Grid();
$myGrid->tabla = "ciudades";
$myGrid->FormAsoc = "FCiudades";
$myGrid->AddGridKeys('CiudadID');
$myGrid->AddColumnasGrid('CiudadID', 'ID');
$myGrid->AddColumnasGrid('Ciudad', 'CIUDAD');
$myGrid->AddColumnasGrid('CodigoPostal', 'CODIGOPOSTAL');
$myGrid->AddColumnasGrid('ProvinciaID', 'PROV.ID');
$myGrid->AddColumnasGrid('(Editar)', 'Editar');

$myGrid->AddColumnasSQL('CiudadID', 'ID');
$myGrid->AddColumnasSQL('Ciudad', 'CIUDAD');
$myGrid->AddColumnasSQL('CodigoPostal', 'CODIGOPOSTAL');
$myGrid->AddColumnasSQL('ProvinciaID', 'PROV.ID');

$myGrid->AddFiltro('CiudadID', isset($_GET['CiudadID']) ? $_GET['CiudadID'] : "", INTEGER);
$myGrid->AddFiltro('Ciudad', isset($_GET['Ciudad']) ? $_GET['Ciudad'] : "", VARCHAR);
$myGrid->AddFiltro('CodigoPostal', isset($_GET['CodigoPostal']) ? $_GET['CodigoPostal'] : "", INTEGER);
$myGrid->AddFiltro('ProvinciaID', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "", INTEGER);
...
...
?>
```

Figura IV.7 - TConCiudades.php

Los anteriores, corresponden a los fragmentos de código de cada uno de los archivos que se necesitan modificar, para reflejar en la aplicación las necesidades del nuevo atributo *CodigoPostal* incorporado; y se resaltan en negrita, las instrucciones (puntos calientes) a ser agregados.

Conclusiones Finales

Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web

Conclusiones finales

El desarrollo del framework propuesto en el presente trabajo, se realizó sobre la base de tres puntos fundamentales:

- Patrones de Diseño. Cómo soluciones estandarizadas y probadas a problemas comunes de desarrollo orientado a objetos, ofreciendo mejores prácticas en torno a los problemas a resolver, favoreciendo la escalabilidad del framework.
- Patrón de arquitectura MVC. Para desacoplar las aplicaciones generadas en tres capas claramente definidas:
  - ✓ Modelo: capa de persistencia y requerimientos de negocio específicos para cada aplicación.
  - ✓ Vista: capa de presentación de las interfaces de usuario
  - ✓ Control: núcleo ajax encargado de la gestión de las peticiones y respuestas entre el cliente y el servidor
- Patrones de interface de usuario. Para el diseño estandarizado de interfaces simples, fáciles utilizar, navegar y recordar, que permitan a los usuarios realizar su trabajo de manera rápida y eficiente.

El framework obtenido, permitió generar de forma rápida y sencilla *-mediante la particularización de sus puntos calientes (hot-spots)-* aplicaciones Web ágiles, amigables y fáciles de mantener, las cuales heredaron naturalmente las características mencionadas en los tres puntos anteriores.

En cuanto a los costos de tiempo y esfuerzo en el desarrollo de aplicaciones a partir del framework, los valores reales resultaron considerablemente más bajos que los estimados mediante los métodos COCOMO y COCOMO II.

Se evaluaron además, la usabilidad y la facilidad de mantenimiento respecto de las aplicaciones extendidas, obteniéndose en cada caso resultados positivos.

El framework desarrollado fue puesto a consideración y uso de un grupo de programadores con niveles intermedios de conocimiento en el desarrollo de

aplicaciones Web, obteniendo como resultado general, un grado de aceptación relativamente bueno.

Por último, y en función de lo expuesto anteriormente, se puede concluir que el framework desarrollado cumple de manera satisfactoria con los objetivos planteados, quedando abierta para futuras investigaciones, incorporar características más avanzadas tales como generadores de código, analizadores sintácticos, e interfaces gráficas que asistan a los programadores en la tarea de desarrollo.

Referencias

[1] **Brooke John,** *"SUS - A quick and dirty usability scale",* http://www.usabilitynet.org/trump/documents/suschapt.doc, Fecha de acceso: 17/12/2008

[2] **CHARTE OJEDA FRANCISCO. (2004),** *"Programación PHP5"*, **Ed. Anaya Multimedia**

[3] **DARIE CRISTIAN, BRINZAREA BOGDAN, CHERECHES-TOSA FILIP, BUCICA MIHAI,** *"Building Responsive Web Applications – AJAX and PHP"*, March 2006, **Ed. Packt Publishing**

[4] **FERRÉ GRAU XAVIER, SÁNCHEZ SEGURA MARÍA ISABEL.** *"Desarrollo Orientado a Objetos"*, ITBA (Instituto Tecnológico de Buenos Aires), Fecha de acceso: 11/07/07

[5] **GAMMA ERICH, HELM RICHARD, JOHNSON RALPH, VLISSIDES JOHN,** *"Patrones De Diseño"*, **Ed. Pearson Educacion**

[6] **GARRETT JESSE JAMES. (2005).** *"Ajax: A New Approach to Web Applications"*, http://www.adaptivepath.com/publications/essays/archives/000385.php, Fecha de acceso: 01/07/07

[7] **GONZÁLEZ RUIZ FRANCISCO, DE LA FUENTE MOYA ANTONIO.** *"COCOMO V2. Modelo de Estimación de Costes para proyectos software",* http://sunset.usc.edu/research/COCOMOII/cocomo_main.html, Fecha de acceso: 11/07/07

[8] **GOLDFARB CHARLES F. & PRESCOD PAUL. (1999)**, *"Manual de XML"*, **Ed. Prentice may**

[9] **GRUPO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE, 2005,** *"Experiencia Y Retos En La Adopción De Frameworks Para El Desarrollo Acelerado De Aplicaciones",* http://www.eafit.edu.co/NR/rdonlyres/E12D52F4-E868-4C79-98C2-B917EADA7892/0/Frameworks.pdf, Fecha de acceso: 25/06/07

[10] **HERNÁNDEZ ELENA, ALVAREZ CARRIÓN GUILLERMO, MUÑOZ ARTEAGA JAIME. (2003).** *"Patrones de Interacción para el Diseño de Interfaces WEB usables"*, http://ccc.inaoep.mx/~grodrig/Descargas/com10017.pdf, Fecha de acceso: 27/06/07

[11] **JACOBSON. I., BOOCH G., RUMBAUGH J. (1999).** *"El proceso unificado de desarrollo de Software"*. **Ed. Addison Wesley**

[12] **JACOBSON. I., BOOCH G., RUMBAUGH J. (1999).** *"Lenguaje de Modelado Unificado"*. **Ed. Addison Wesley**

[13] **MARKIEWICZ MARCUS EDUARDO**, *"El Desarrollo del Framework Orientado al Objeto"*, http://www.acm.org/crossroads/espanol/xrds7-4/frameworks.html, Fecha de Acceso: 28/06/07

[14] **MARTÍNEZ JOSÉ SÁEZ, GARCÍA MOLINA JESÚS, JIMÉNEZ GARCÍA PEDRO J.,** *"Una Arquitectura para una Herramienta de Patrones de Diseño",* http://dis.um.es/~jmolina/arquipatronesjis99.pdf, Fecha de acceso: 20/06/07

[15] **MORENO M. ANA, SÁNCHEZ-SEGURA MARIBEL. (2003).** *"Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico",* http://www.willydev.net/descargas/prev/PatronesUsa.pdf, Fecha de acceso: 28/06/07

[16] **MORENO M. ANA, CAPUCHINO S.** *"COCOMO II. Estimación de Proyectos Software",* http://www.inf.uach.cl/rvega/asignaturas/info265/cocomoii.pdf, Fecha de acceso: 11/07/07

[17] **MATEO MANUEL PALACÍN. (2007).** *"Portal WEB 2.0 utilizando Framework Struts"*, https://upcommons.upc.edu/pfc/bitstream/2099.1/4043/1/memoria.pdf, Fecha de acceso: 20/06/07

[18] **PHP.NET.** *"Documentación PHP 5 Orientado a Objetos"*, http://www.php.net/manual/es/language.oop5.php, Fecha de acceso: 11/07/07

[19] **PHP.NET.** *"Sitio Oficial de PHP"*, http://www.php.net, Fecha de acceso: 11/07/07

[20] **PRESSMAN, ROGER. (2002),** *"Ingeniería del Software – Un enfoque práctico",* 5ta Edición 2002. **Ed. McGraw Hill**

[21] **VAN WELIE MARTIN, VAN DER VEER GERRIT C., ELIËNS ANTON. (2000).** *"Patterns as Tools for User Interface Design",* http://www.cs.vu.nl/~martijn/gta/docs/TWG2000.pdf, Fecha de acceso: 28/06/07

[22] **WELICKI LEÓN,** *"Patrones y Antipatrones: una Introducción - Parte II",* http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3317.asp Fecha de acceso: 20/06/07

[23] **WIKIPEDIA.** *"Definición de Framework"*, http://es.wikipedia.org/wiki/Framework, Fecha de acceso: 04/07/07

Anexo A
Caso Práctico

Arquitectura software reutilizable basada en patrones de diseño y patrones de interacción, para el desarrollo rápido de aplicaciones web

ANEXO A

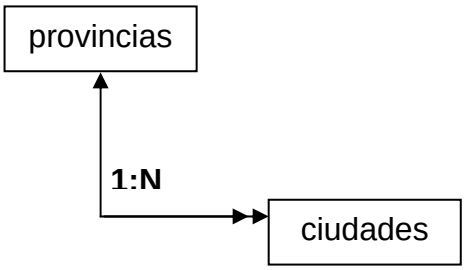
A continuación se presenta un caso práctico de uso del framework propuesto. Si bien se trata de un caso de aplicación sencillo, ayudará a entender el sentido del presente trabajo. Se asume que los requisitos funcionales de la aplicación a extender, son los mismos que los indicados en la sección IV.3.1

1. Estructura de la Base de Datos

Se dispone de una base de datos con dos tablas relacionadas, *provincias* y *ciudades*, con la siguiente estructura:

provincias	
Atributo	**Tipo**
ProvinciaID **(*)**	Autoincremental
Provincia	Texto

ciudades	
Atributo	**Tipo**
CiudadID **(*)**	Autoincremental
Ciudad	Texto
ProvinciaID	Entero



(*) Atributos Clave.

2. Generación de la aplicación

Para cada tabla, se crean con ayuda de las plantillas descriptas en el punto 3 de la sección III.4.3, los archivos que generan los patrones de interface, de acuerdo al criterio indicado en el punto 2 de la misma sección, según sea necesario. Esto es:

Tabla	Archivo	Plantilla	Propósito
provincias	TConProvincias.php	Maestro de Registros	Maestro de registros de provincias
	FProvincias.php	Formulario	Alta, baja y actualización de registros
	PProvincias.php	Lógica de Negocio	Lógica de negocios
	VGralProvincias.php	Vista General	Modela la navegación a través de la relación 1:N entre provincias y ciudades

Tabla	Archivo	Plantilla	Propósito
ciudades	TConCiudades.php	Maestro de Registros	Maestro de registros de ciudades
	FCiudades.php	Formulario	Alta, baja y actualización de registros
	PCiudades.php	Lógica de Negocio	Lógica de negocios

A continuación se muestra el código para cada uno de los archivos, particularizado al caso de uso práctico.

Patrón de interacción *"Maestro de Registros"* asociado a la tabla provincias.

```
<?php
session_start();
$_SESSION['PageReturn'] = 'loadTCPage("'.$_SERVER['PHP_SELF'].'?'.$_SERVER['QUERY_STRING'].'",
"PlaceHolder")';
$_SESSION['PageReturnVG'] = '';

include("clases/PatternFactory.php");
include("clases/Grid.php");
include("clases/Form.php");
$page = $_GET["page"];

$ObjTC = new TrabajarConFactory(); A) Se instancia la clase TrabajarConFactory

$myForm = $ObjTC->get_Filtro(); B) Se configura un Objeto Formulario para filtro de datos
$myForm->tabla = "provincias";
$myForm->modo = "INS";
$myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROV.ID.', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddAtributos('Provincia', VARCHAR, 30, 'PROVINCIA', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddDefaultValues('ProvinciaID',  isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "");
$myForm->AddDefaultValues('Provincia',  isset($_GET['Provincia']) ? $_GET['Provincia'] : "");
$myForm->readForm();

$myGrid = $ObjTC->get_Grid(); C) Se configura un Objeto Grilla para el maestro de registros
$myGrid->tabla = "provincias";
$myGrid->FormAsoc = "FProvincias";
$myGrid->AddGridKeys('ProvinciaID');
$myGrid->AddColumnasGrid('ProvinciaID', 'ID');
$myGrid->AddColumnasGrid('Provincia', 'PROVINCIA');
$myGrid->AddColumnasGrid('(Editar)', 'Editar');
$myGrid->AddColumnasGrid('(Borrar)', 'Borrar');
$myGrid->AddColumnasSQL('ProvinciaID', '');
$myGrid->AddColumnasSQL('Provincia', '');
$myGrid->AddFiltro('ProvinciaID', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "", INTEGER);
$myGrid->AddFiltro('Provincia', isset($_GET['Provincia']) ? $_GET['Provincia'] : "", VARCHAR);

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; D) Se genera la salida en formato XML
echo '<data>';
echo '<titulo>Trabajar con Provincias</titulo>';
echo '<Caller>TCPage</Caller>';
echo '<callerNamePag>TConProvincias</callerNamePag>';
echo '<NameTC>TConProvincias</NameTC>';
echo '<NameForm>FProvincias</NameForm>';
echo $myForm->getFormXML();
echo $myGrid->getGridXML("", "", $page);
echo $myGrid->getVistasXML('ProvinciaID', 'VGralProvincias');
echo '</data>';
?>
```

TConProvincias.php

Se resaltan en negrita 4 puntos importantes:

A) Instancia de la clase TrabajarConFactory. Se crea una instancia de la clase encargada de la gestión y creación de los diferentes patrones de interface.

B) Configuración del formulario para filtro de datos.

- Mediante el comando *AddAtributos*, se especifican los atributos a incluir en el formulario de filtro
- *AddDefaultValues*, permite mantener el valor de cada atributo en cada filtro o recuperación de datos
- La propiedad *"tabla"*, define a provincias como tabla asociada al formulario de filtros

C) Configuración de la grilla para el maestro de registros

- La propiedad *"tabla"*, define a *provincias* como tabla asociada a la grilla
- *"FormAsoc"*, define el nombre del script correspondiente al formulario de actualización asociado a la grilla (FPovincias.php)
- *AddGridKeys*, permite definir el(las) clave(s) asociadas a la tabla
- *AddColumnasGrid*, permite definir los atributos para las columnas de la grilla
- *AddColumnasSQL*, define el conjunto de atributos con los que se genera la consulta SQL. En general, coincide con los atributos indicados mediante *AddColumnasGrid*
- *AddFiltro*, permite establecer un valor para un determinado atributo en la consulta SQL, en función de los valores introducidos mediante el formulario de filtros

D) Salida de la especificación XML

- Mediante *getVistasXML*, se define un punto de acceso a partir del atributo ProvinciaID, a la vista general "VGralProvincias.php" correspondiente a un registro.
- Los valores de las etiquetas <*callerNamePag*> y <*NameTC*> se completan con el nombre del script TConProvincias.php; y el valor de <*NameForm*>, con el nombre del script asociado al formulario de actualización, FProvincias.php.

Patrón de interacción *"Formulario"* asociado a la tabla provincias.

```
<?php
include("clases/PatternFactory.php");
include("clases/Form.php");

$mod = isset($_GET['Mod']) ? $_GET['Mod'] : "";
switch($mod){
        case 'INS': $titulo = "Nuevo Registro"; break;
        case 'UPD': $titulo = "Modificar Registro"; break;
        case 'DLT': $titulo = "Confirma borrar Registro ?"; break;
        default: $titulo = "Error. Comando desconocido...";exit;
}

$ ObjForm = new TrabajarConFactory(); A) Se instancia la clase TrabajarConFactory

$myForm = $ObjForm->get_Form(); B) Se configura un Objeto Formulario
$myForm->tabla = "provincias";
$myForm->modo = $mod;
$myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROV.ID', 'FALSE','TEXTFIELD', 'visible');
$myForm->AddAtributos('Provincia', VARCHAR, 70, 'PROVINCIA', 'TRUE','TEXTFIELD', 'visible');
$myForm->AddAtributosClave("ProvinciaID", isset($_GET["ProvinciaID"]) ? $_GET["ProvinciaID"] : "");
$myForm->MakeFiltro();
$myForm->readForm();

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; C) Se genera la salida en formato XML
echo '<data>';
echo '<titulo>'.$titulo.'</titulo>';
echo '<NameTC>TConProvincias</NameTC>';
echo '<NameForm>FProvincias</NameForm>';
echo '<NameP>PProvincias</NameP>';
echo '<modo>'.$mod.'</modo>';
echo ($myForm->getFormXML($mod);
echo '</data>';
?>
```

FProvincias.php

Se resaltan en negrita 3 puntos importantes:

A) Instancia de la clase TrabajarConFactory.

B) Configuración del formulario.

- Mediante el comando *AddAtributos*, se especifican los atributos a incorporar en el formulario de filtro
- *AddAtributosClave*, permite definir el(los) atributos claves de la tabla provincias asociadas
- La propiedad *"tabla"*, define a provincias, como tabla asociada al formulario de filtros

C) Salida de la especificación XML. Los valores de las etiquetas *<NameTC>*, *<NameForm>* y *<NameP>* se completan con los nombres de los script TConProvincias.php, FProvincias.php, y Provincias.php respectivamente.

Patrón de interacción *"Lógica de Negocios"* asociado a la tabla provincias.

```
<?php
session_start();
require_once('clases/Form.php');
require_once('clases/MyADO.php');
require_once('clases/Observer.php');
require_once('config.php');

function BeforeInsert(){ }
function BeforeUpdate(){ }
function BeforeDelete(){ }
function AfterInsert($ID_Insert){ }
function AfterUpdate($ID_Update){ }
function AfterDelete(){ }

$mod = $_GET['Mod'];
if($mod!="cancel" && $mod!="cancelVG"){
        $Ado = new MyADO($mod); A) Se instancia la clase MyADO
        $Ado->tablaBase = "provincias";
        $Ado->getAttrTipos();
        $Ado->AddAtributosClave("ProvinciaID", $_REQUEST["ProvinciaID"]);

        switch($mod){ B)
                case 'INS': $Ado->AddAtributosInsert($_REQUEST, "ProvinciaID;Provincia"); break;
                case 'UPD':
                $Ado->AddAtributosUpdate($_REQUEST, "ProvinciaID;Provincia"); break;
                case 'DLT': break;
                default:echo 'Server error: client command missing.'; exit;
        }

        if ($Ado->MySQLExec()){
                if($_SESSION["PageReturnVG"]=='') echo $_SESSION["PageReturn"];
                else echo $_SESSION["PageReturnVG"];
        }else{
                $Ado->rollback();
                 echo "Err";
        }
}else{
        if($_SESSION["PageReturnVG"]=='' || $mod=='cancelVG') echo $_SESSION["PageReturn"];
        else echo $_SESSION["PageReturnVG"];
}
?>
```

PProvincias.php

Se resaltan en negrita 2 puntos importantes:

- **A) Instancia de la clase MyADO.** Se crea una instancia de la clase encargada de la gestión y conexión con la base de datos.
- **B)** Los comandos *AddAtributosInsert* y *AddAtributosUpdate*, permiten definir el conunto de atributos a actualizar pasados a través de un formulario.

Patrón de interacción *"Vista General"* asociado a la tabla provincias.

```
<?php
session_start();
$_SESSION['PageReturnVG'] = 'loadVistaGral("'.$_SERVER['PHP_SELF'].'?'.$_SERVER['QUERY_STRING'].'",
''PlaceHolder'')';

include("clases/PatternFactory.php");
include("clases/Grid.php");
```

VGralProvincias.php

```
include("clases/Form.php");
include("clases/Tabs.php");

$tab = isset($_GET['Tab']) ? $_GET['Tab'] : "";
$pag = isset($_GET['page']) ? $_GET['page'] : "1";

$ObjVista = new TrabajarConFactory(); A) Se instancia la clase TrabajarConFactory

$myTab = $ObjVista->get_Tabs();B) Se configura un Objeto Tabs
$myTab->SelTab = $tab;
$myTab->AddTabs("General:Ciudades");
$myTab->AddKeysTabs("ProvinciaID", isset($_GET["ProvinciaID"]) ? $_GET["ProvinciaID"] : "");

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; C) Se genera la salida en formato XML
echo '<data>';
echo '<Caller>TCPageVtaGral</Caller>';
echo '<callerNamePag>VGralProvincias</callerNamePag>';
echo '<NameP>PProvincias</NameP>';
echo '<NameForm>FProvincias</NameForm>';
echo '<TabActivo>'.$myTab->SelTab.'</TabActivo>';
echo '<VtaGral>'.$myTab->getTabsXML().'</VtaGral>';

switch($tab){
        case "General": D) Se configura un Objeto Formulario
                $myForm = $ObjVista ->get_Form();
                $myForm->tabla = "provincias";
                $myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROV.ID.', 'FALSE', 'LABEL', 'visible');
                $myForm->AddAtributos('Provincia', VARCHAR, 50, 'PROV.', 'FALSE', 'LABEL', 'visible');
                $myForm->AddAtributosClave("ProvinciaID", isset($_GET["ProvinciaID"]) ? $_GET["ProvinciaID"] :
"");
                $myForm->MakeFiltro();
                $myForm->readForm();
                echo $myForm->getFormXML('DSP');
        break;
        case "Ciudades": E) Se configura un Objeto Grilla para la vista asociada
                $myGrid = $ObjVista ->get_Grid();
                $myGrid->tabla = "ciudades";
                $myGrid->AddGridKeys('CiudadID');
                $myGrid->FormAsoc = "FCiudades";
                $myGrid->AddColumnasGrid('CiudadID', 'ID');
                $myGrid->AddColumnasGrid('Ciudad', 'CIUDAD');
                $myGrid->AddColumnasGrid('(Editar)', 'Editar');
                $myGrid->AddColumnasGrid('(Borrar)', 'Borrar');
                $myGrid->AddColumnasSQL('CiudadID', 'ID');
                $myGrid->AddColumnasSQL('Ciudad', 'CIUDAD');
                $myGrid->AddForeingKeys('ProvinciaID', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "",
INTEGER);
                echo $myGrid->getGridXML("", "", $pag);
        break;
}
echo '</data>';
?>
```

VGralProvincias.php (continuación)

Se resaltan en negrita 5 puntos importantes:

A) Instancia de la clase TrabajarConFactory.

B) Configuración de los diferentes puntos de acceso a las vistas.

- La propiedad *SelTab*, permite establecer la vista que se encuentra activa
- *AddTabs*, permite definir las diferentes vistas del patrón
- *"AddKeyTabs"*, define el(los) atributos clave para la vista general. Y se corresponde con el(las) claves de la tabla asociada.

C) Salida de la especificación XML

- Los valores de las etiquetas *<callerNamePag>*, *<NameP>* y *<NameForm>*, se completan con los nombres de los script correspondientes a VGralProvincias.php, PProvincias.php, y FProvincias.php respectivamente.

D) Configuración del Formulario General. El procedimiento es idéntico al de los casos de configuración de formularios, explicados anteriormente.

E) Configuración de la grilla para una vista asociada. El procedimiento es idéntico al de los casos de configuración de grillas, explicados anteriormente.

Del mismo modo antes descripto, se procede a la creación de los patrones correspondientes a la tabla ciudades:

```
<?php
session_start();
$_SESSION['PageReturn'] = 'loadTCPage("'.$_SERVER['PHP_SELF'].'?'.$_SERVER['QUERY_STRING'].'",
"PlaceHolder")';
$_SESSION['PageReturnVG'] = '';

include("clases/PatternFactory.php");
include("clases/Grid.php");
include("clases/Form.php");

$action = $_GET['action'];
$page = $_GET["page"];
$ObjTC = new TrabajarConFactory();; A) Se instancia la clase TrabajarConFactory

$myForm = $ObjTC->get_Filtro();B) Se configura un Objeto Formulario para filtro de datos
$myForm->tabla = "ciudades";
$myForm->modo = "INS";
$myForm->AddAtributos('CiudadID', INTEGER, 5, 'CIUDAD.ID.', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddAtributos('Ciudad', VARCHAR, 30, 'CIUDAD', 'TRUE', 'TEXTFIELD', 'visible');
$myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROVINCIA', 'TRUE', 'DBComboBox', 'visible');

$myForm->AddDefaultValues('CiudadID',  isset($_GET['CiudadID']) ? $_GET['CiudadID'] : "");
$myForm->AddDefaultValues('Ciudad',  isset($_GET['Ciudad']) ? $_GET['Ciudad'] : "");
$myForm->AddDefaultValues('ProvinciaID',  isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "");

$myForm->setTipoDBComboBox('ProvinciaID',  'Provincia', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "",
'provincias');

$myForm->readForm();
$myGrid = $ObjTC->get_Grid();C) Se configura un Objeto Grilla para el maestro de registros
$myGrid->tabla = "ciudades";
$myGrid->FormAsoc = "FCiudades";
$myGrid->AddGridKeys('CiudadID');
$myGrid->AddColumnasGrid('CiudadID', 'ID');
$myGrid->AddColumnasGrid('Ciudad', 'CIUDAD');
$myGrid->AddColumnasGrid('ProvinciaID', 'PROV.ID');
$myGrid->AddColumnasGrid('(Editar)', 'Editar');

$myGrid->AddColumnasSQL('CiudadID', 'ID');
$myGrid->AddColumnasSQL('Ciudad', 'CIUDAD');
$myGrid->AddColumnasSQL('ProvinciaID', 'PROV.ID');
```

TConCiudades.php

```
$myGrid->AddFiltro('CiudadID', isset($_GET['CiudadID']) ? $_GET['CiudadID'] : "", INTEGER);
$myGrid->AddFiltro('Ciudad', isset($_GET['Ciudad']) ? $_GET['Ciudad'] : "", VARCHAR);
$myGrid->AddFiltro('ProvinciaID', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "", INTEGER);

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; D) Se genera la salida en formato XML
echo '<data>';
echo '<titulo>Trabajar con Ciudades</titulo>';
echo '<Caller>TCPage</Caller>';
echo '<callerNamePag>TConCiudades</callerNamePag>';
echo '<NameTC>TConCiudades</NameTC>';
echo '<NameForm>FCiudades</NameForm>';
echo utf8_decode($myForm->getFormXML());
echo utf8_decode($myGrid->getGridXML("", "", $page));
echo '</data>';
?>
```

TConCiudades.php (continuación)

```
<?php
include("clases/PatternFactory.php");
include("clases/Form.php");
$mod = isset($_GET['Mod']) ? $_GET['Mod'] : "";
switch($mod){
        case 'INS': $titulo = "Nuevo Registro";break;
        case 'UPD': $titulo = "Modificar Registro"; break;
        case 'DLT': $titulo = "Confirma borrar Registro ?"; break;
        default: $titulo = "Error. Comando desconocido..."; exit;
}

$ObjForm = new TrabajarConFactory(); A) Se instancia la clase TrabajarConFactory

$myForm = $ObjForm->get_Form();B) Se configura un Objeto Formulario
$myForm->tabla = "ciudades";
$myForm->modo = $mod;
$myForm->AddAtributos('CiudadID', INTEGER, 5, 'CIUDAD.ID', 'FALSE','TEXTFIELD', 'visible');
$myForm->AddAtributos('Ciudad', VARCHAR, 70, 'CIUDAD', 'TRUE','TEXTFIELD', 'visible');
$myForm->AddAtributos('ProvinciaID', INTEGER, 5, 'PROV.ID', 'TRUE','DBComboBox', 'visible');

$myForm->SetTipoDBComboBox("ProvinciaID", "Provincia", isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "",
"provincias");
$myForm->AddDefaultValues('ProvinciaID', isset($_GET['ProvinciaID']) ? $_GET['ProvinciaID'] : "");
$myForm->AddAtributosClave("CiudadID", isset($_GET["CiudadID"]) ? $_GET["CiudadID"] : "");
$myForm->MakeFiltro();
$myForm->readForm();

echo '<?xml version="1.0" encoding="ISO-8859-1"?>'; C) Se genera la salida en formato XML
echo '<data>';
echo '<titulo>'.$titulo.'</titulo>';
echo '<NameTC>TConCiudades</NameTC>';
echo '<NameForm>FCiudades</NameForm>';
echo '<NameP>PCiudades</NameP>';
echo '<modo>'.$mod.'</modo>';
//Retorna las filas de la grilla
echo utf8_decode($myForm->getFormXML($mod));
echo '</data>';
?>
```

FCiudades.php

```
<?php
session_start();
require_once('clases/Form.php');
require_once('clases/Observer.php');
require_once('clases/MyADO.php');
require_once('config.php');

function BeforeInsert(){ }
function BeforeUpdate(){ }
function BeforeDelete(){ }
function AfterInsert($ID_Insert){ }
function AfterUpdate($ID_Update){ }
function AfterDelete(){ }

if (!isset($_GET['Mod'])){  //Lee la acción a realizar
  echo 'Server error: client command missing.';  exit;
}else{
        $mod = $_GET['Mod'];
        if($mod!="cancel" && $mod!="cancelVG"){
                $Ado = new MyADO($mod); A) Se instancia la clase MyADO
                $Ado->tablaBase = "ciudades";
                $Ado->getAttrTipos();
                $Ado->AddAtributosClave("CiudadID", $_REQUEST["CiudadID"]);
                switch($mod){ B)
                        case 'INS': $Ado->AddAtributosInsert($_REQUEST, "CiudadID;Ciudad;ProvinciaID");
                        break;
                        case 'UPD':
                        $Ado->AddAtributosUpdate($_REQUEST, "CiudadID;Ciudad;ProvinciaID");
                        break;
                        case 'DLT':
                        break;
                        default:echo 'Server error: client command missing.';
                        exit;
                }

                if ($Ado->MySQLExec()){
                        if($_SESSION["PageReturnVG"]=='') echo $_SESSION["PageReturn"];
                        else echo $_SESSION["PageReturnVG"];
                }else{
                        $Ado->rollback();
                         echo "Err";
                }
        }else{
                if($_SESSION["PageReturnVG"]=='' || $mod=='cancelVG') echo $_SESSION["PageReturn"];
                else echo $_SESSION["PageReturnVG"];
        }
}
?>
```

PCiudades.php

Resulta importante destacar que el código generado, no debe modificar la estructura fundamental de las plantillas. Estas deben ser completadas y configuradas solo en sus puntos parametrizables haciendo uso de los puntos calientes indicados en las secciones III.3.5.1 y III.4.2.

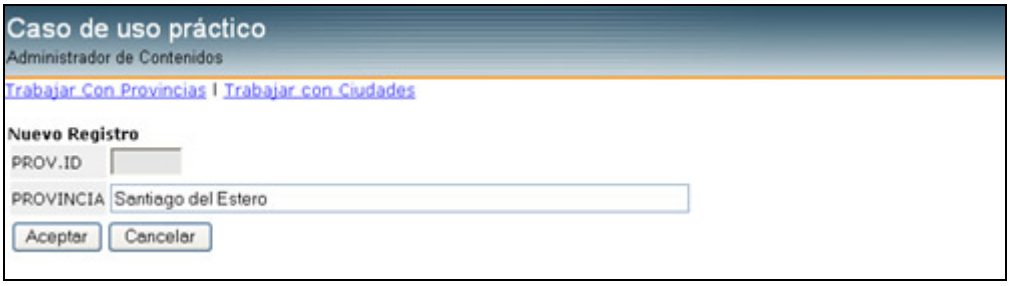
Por ultimo, y luego de generar todos los archivos necesarios para cada una de las tablas a mantener, se obtiene una aplicación Web funcional y sencilla, de acuerdo a los requerimientos indicados en la sección IV.3.1.

TConProvincias.php

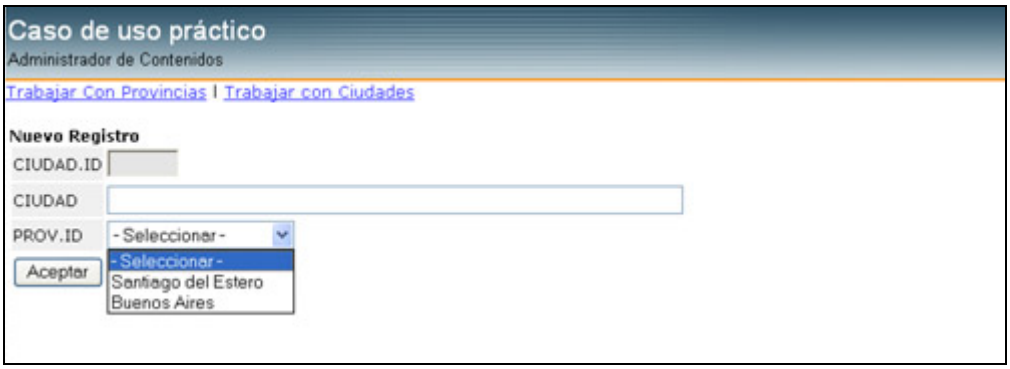
Las pantallas Maestro de Registros, representan un punto de inicio de las aplicaciones extendidas del framework propuesto. En este caso particular, es posible filtrar datos por Código y Nombre de Provincia. Además, el patrón da la posibilidad de editar, eliminar o dar de alta un nuevo registro; y de llamar a la vista general del mismo, por medio de un link en cada valor del atributo de la primer columna de la grilla.

VGralProvincias.php

Una vista general, proporciona una visión general de un registro, y el conjunto de de sus registros asociados. Pensemos en el caso de un artículo periodístico, el cual puede tener asociados archivos de imagen, audio y video. El camino más corto para determinar el estado de archivos asociados al artículo, resulta sin dudas una vista general del registro.



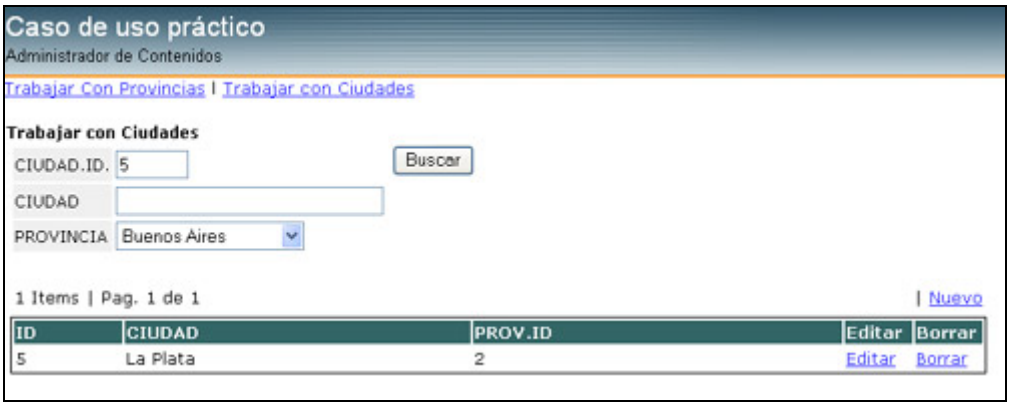
FProvincias.php



FCiudades.php

Los formularios son patrones que se comportan en función al modo con el que son convocados. Mientras que en caso de las pantallas *"Maestro de registros"* y *"Vista General"*, sirven únicamente para filtro y representación de datos, cuando son llamados en modo inserción, modificación o eliminación, permiten ingresar, modificar y eliminar registros.

TConCiudades.php



TConCiudades.php