UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: VF Framework: Framework para la incorporación de funcionalidades volátiles
Autores: Darián Frajberg
Director: Dr. Gustavo Rossi
Codirector: Dr. Matías Urbieta
Asesor profesional:
Carrera: Licenciatura en Sistemas

Resumen

La mayoría de las aplicaciones web de hoy en día están caracterizadas, sin lugar a dudas, por su gran dinamismo y su continua evolución. Luego de implementarse y efectuarse el primer deployment de una aplicación web, suelen surgir nuevos requerimientos que implican la necesidad de incorporar nuevas funcionalidades, generalmente desconocidas durante la etapa de diseño. Algunas veces, estas funcionalidades son probadas por un tiempo determinado y luego son descartadas por no haber resultado lo suficientemente útiles para los usuarios. Otras veces, aparecen como respuesta a determinados eventos y/o condiciones. Por último, es muy habitual que ciertas funcionalidades deban ser activadas periódicamente en determinado momento del año, para luego ser desactivadas.
La continua incorporación y remoción de estas funcionalidades, que llamaremos “funcionalidades volátiles”, usualmente impacta de manera negativa en importantes aspectos de las aplicaciones web.
VF Framework es un marco de trabajo conceptual y tecnológico que permite mejorar el ciclo de vida de las funcionalidades volátiles en las aplicaciones web. Este objetivo es logrado a partir de dos principios fundamentales que son: el desacoplamiento de las funcionalidades volátiles de la aplicación original; y la posibilidad de programar la activación y desactivación de dichas funcionalidades volátiles.

Palabras Claves

Funcionalidad Volátil, Aplicación Web, Enfoque, VF Framework, Cambio, Modificación de capas de aplicación, Desacoplamiento, Programación de eventos

Conclusiones

Como conclusión, resulta clara la necesidad actual de contar con alguna alternativa que permita mejorar la manipulación de aplicaciones web ante la aparición de funcionalidades volátiles.
El VF Framework efectivamente provee soluciones para dicho problema tan recurrente. El mismo ha sido llevado a la práctica y probado de manera satisfactoria mediante la implementación de prototipos. Además, el enfoque ha sido presentado a un grupo de desarrolladores imparciales que ha tenido como respuesta una muy buena aceptación.

Trabajos Realizados

Se ha realizado la investigación de los temas base de la tesina de grado propuesta.
Se han analizado y definido los diferentes requerimientos con los que un lenguaje de programación debería contar para poder implementar el VF Framework.
Se han implementado satisfactoriamente prototipos para los lenguajes de programación Java y Smalltalk.
Se ha realizado un experimento para evaluar y obtener feed-back sobre el marco de trabajo presentado.

Trabajos Futuros

Como trabajo futuro para la investigación, restaría la realización de estudios y evaluaciones de impacto que permitieran medir empíricamente cuánto afecta a una aplicación web la aparición de funcionalidades volátiles, y compararlo con lo que ocurre al utilizar el framework propuesto.
Además, se debería avanzar con el desarrollo del enfoque, profundizando los conceptos investigados y mejorando y generando nuevas implementaciones sólidas y maduras que puedan ser aplicados en la industria por la comunidad.

Fecha de la presentación: Mayo del 2015

“VF Framework”: Framework para la incorporación de funcionalidades volátiles

Darián Frajberg[1]

Trabajo final para obtener el grado de

Licenciado en Sistemas / Licenciatura en Sistemas

De la
Facultad de Informática,
Universidad Nacional de La Plata,
Argentina

Director: Dr. Gustavo Rossi
Codirector: Dr. Matías Urbieta

La Plata, Mayo del 2015

[1] Nro. Alumno: 10250/6 – darian_f@hotmail.com

Índice General

Índice de ilustraciones

1 Agradecimientos

En primer lugar, quiero agradecerle a toda mi familia por el gran apoyo que me ha brindado a lo largo de todos estos años de estudio y de toda mi vida.
A mi mamá, Fabiana, y a mi papá, Marcelo, a quienes les debo gran parte de este logro.
A mi hermana, Eliana, con quien tanto me divierto día a día.
A mis abuelos, siempre pendientes de mí y de todo lo que hago.
A mis amigos y todas esas personas que de una u otra manera siempre estuvieron presentes.
Por último, quiero agradecerles inmensamente a mi director, Gustavo, y a mi codirector, Matías. Su apoyo, su confianza y su capacidad para guiarme han sido aportes invaluables, no solamente para el desarrollo de esta tesis, sino también para mi formación y mi persona.

Darián

2 Introducción

2.1 Introducción y motivación

Una de las características principales de la mayoría de las aplicaciones web de hoy en día es su gran dinamismo. Sin lugar a dudas, están caracterizadas por una continua evolución. Luego de implementarse y efectuarse el primer deployment de una aplicación web, suelen surgir nuevos requerimientos que implican la necesidad de incorporar nuevas funcionalidades, generalmente desconocidas durante la etapa de diseño.

Algunas veces, estas funcionalidades son probadas por un tiempo determinado y luego son descartadas por no haber resultado lo suficientemente útiles para los usuarios. Otras veces, aparecen como respuesta a determinados eventos y/o condiciones. Por último, es muy habitual que ciertas funcionalidades deban ser activadas periódicamente en determinado momento del año, para luego ser desactivadas. Es decir, son introducidas y eliminadas de una aplicación reiteradamente.

Este tipo de funcionalidades, que surgen como respuesta a requerimientos inesperados de la capa de negocio, tienen la particularidad de que eventualmente deben ser removidos debido a la caducidad de su valor comercial.

La continua incorporación y remoción de estas funcionalidades, que llamaremos "funcionalidades volátiles", usualmente impacta de manera negativa en algunos aspectos importantes de la aplicación Web. Esto es así dado que esta tarea se realiza de manera manual y es bastante propensa a errores. Por consiguiente, tanto la calidad del código de la aplicación, como el funcionamiento de la misma se ven perjudicados. Además, por supuesto esto conlleva un trabajo extra por parte del equipo de desarrolladores en la etapa de mantenimiento, y consecuentemente un aumento de costos y tiempos para llevar a cabo su tratamiento. Cabe destacar que las funcionalidades volátiles pueden ser sumamente complejas e involucrar cambios en todas las capas de una aplicación (capa de negocio, capa de navegación y capa de presentación).

A continuación se enumeran algunos de los problemas típicos que aparecen ante la remoción de funcionalidades volátiles:

- Problemas en la capa de negocio: variables y métodos no referenciados, fragmentos de código sin utilidad distribuidos a lo largo de la aplicación, código comentado.
- Problemas en la capa de navegación: sevicios inyectados y métodos no referenciados, métodos sin comportamiento.
- Problemas en la capa de presentación: elementos HTML introducidos innecesarios (sean percibibles o no), importación innecesarios de librerías y recursos como CSS y código Javascript.
- Problemas en la base de datos: Pérdida de información, generación de datos inválidos.

Adicionalmente, debe considerarse que ante múltiples iteraciones de introducciones y remociones de funcionalidades volátiles, la aplicación va gradualmente perdiendo su calidad cada vez más e incurriendo en la introducción de mayor cantidad de errores.

Un claro ejemplo de esto se da en aplicaciones web de empresas dedicadas a la comercialización de productos y servicios vía online.

Tomemos el caso de Cyber Monday. Cyber Monday es un término de marketing utilizado para referirse al lunes posterior al día de Acción de Gracias en los Estados Unidos. Es una jornada que fue creada por grandes empresas para incentivar la compra en línea a través de grandes descuentos.

Hace algunos años Argentina copió esta iniciativa y los resultados han sido altamente positivos. Tanto es así que a partir del 2014 se ha sumado un nuevo evento de jornadas de descuentos llamado "Hot Sale". Este evento, que se realizará nuevamente durante los días 15, 16 y 17 de Mayo del 2015, contará con más de 180 comercios del país que ya han confirmado su participación.

Como ejemplo, en la Ilustración 1 se muestran las promociones de "Hot Sale" lanzadas por Falabella.com. En el parte superior de la imagen, se muestra un banner, en donde se promociona la compra de productos de electrónica con hasta 12 cuotas sin interés para determinadas tarjetas de crédito. A su vez, en el costado izquierdo de la imagen se anuncia un descuento del 30% para los clientes de determinados bancos. En el medio, puede observarse un carrusel con productos que tienen importantes descuentos. Finalmente, en la parte inferior, se visualiza otro banner con el tiempo restante de promoción y uno de los productos promocionados con su correspondiente descuento. Estos componentes permiten la navegación hacia productos específicos en venta. La customización de Falabella.com para "Hot Sale" no sólo ha requerido la modificación de páginas web mediante la introducción de imágenes y links. También ha sido necesaria la consideración de otras cuestiones tales como la adición de nuevas reglas de negocio específicas para poder efectuar los descuentos promocionados, el registro de las ventas realizadas con sus correspondientes promociones y la definición particular de descuentos para los diferentes productos involucrados, entre otras.

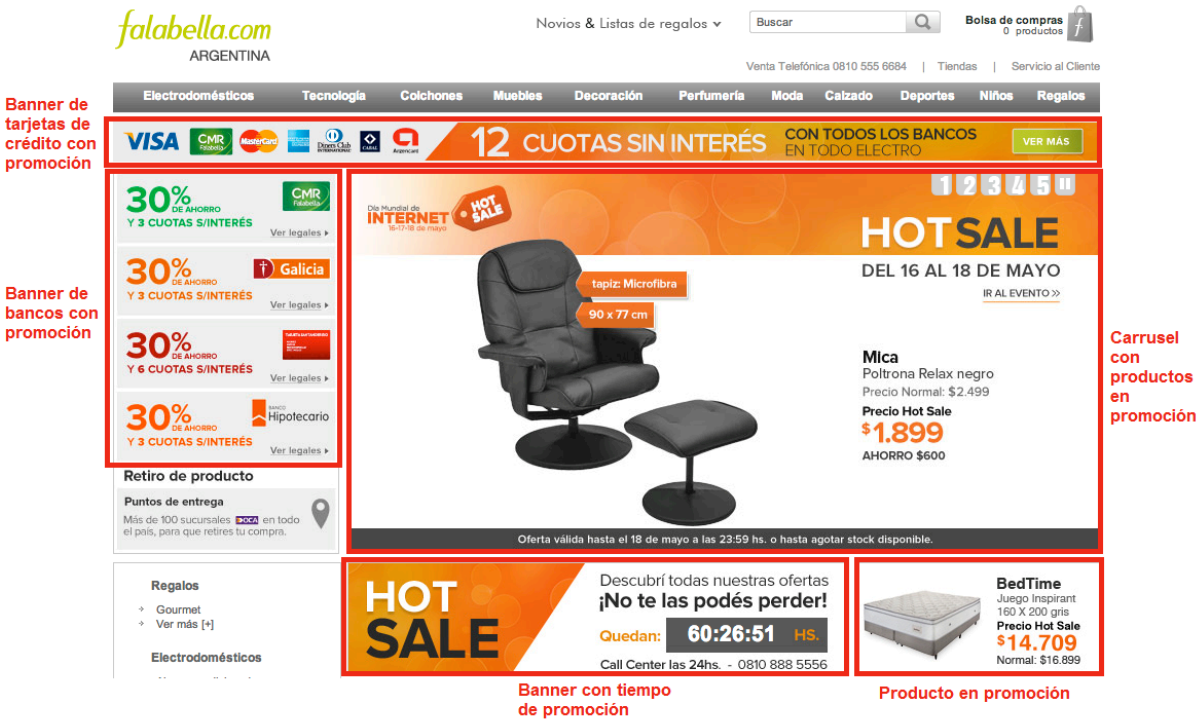


Ilustración 1: Hot Sale en Falabella.com

Este tipo de eventos son cada vez más habituales, por cuanto también debemos considerar otras fechas especiales, en las cuales el lanzamiento de promociones de

productos es muy común. Navidad, el día de la madre, Reyes, y el día de los enamorados son algunos ejemplos de ello.

Como puede observarse, este tipo de situaciones son muy comunes, y dada la velocidad en la que se mueve el mercado, su resolución debería poder ser gestionada de la manera más fácil, rápida y eficiente que sea posible. Sin embargo, en la actualidad, no existe ninguna herramienta de apoyo que permita simplificar y hacerse cargo de estas cuestiones satisfactoriamente.

2.2 Definición de objetivos

Tal como fue discutido en [1], para el ciclo de vida de un proyecto de software, la aparición e introducción de funcionalidades volátiles habitualmente representan un problema.

El objetivo de esta tesis es proveer un marco de trabajo para implementar los conceptos definidos en [1] en aplicaciones web. Para probar el marco de trabajo se desarrollaron dos prototipos. Para el primero de los prototipos se ha utilizado Java, tecnologías de transformación de documentos XML, Programación Orientación a Aspectos y motor de reglas. Para el segundo de los prototipos se ha utilizado Smalltalk.

Este Framework propone el desacoplamiento de este tipo de funcionalidades de la aplicación base. Es decir, la separación de las funcionalidades volátiles de la aplicación original, lográndose así mantener la integridad e independencia de esta última. Para ello es necesario ubicar por un lado el código correspondiente a la aplicación original que permanecerá intacto, y por otro lado todo el código correspondiente a las funcionalidades volátiles que será utilizado para enriquecer la aplicación original. Asimismo, las funcionalidades volátiles a introducir deben ser capaces de modificar todas las capas pertenecientes a la aplicación original (es decir la capa de presentación, la capa de navegación y la capa de negocio).

Una ventaja importante que provee el marco de trabajo propuesto se da al optar por pasar de una versión de la aplicación con determinadas funcionalidades volátiles incorporadas a una versión con menos funcionalidades volátiles o inclusive a la versión original (sin ninguna funcionalidad volátil del todo). El proceso para llevar a cabo este tipo de remociones se ve muy simplificado. Esto es así dado que como la aplicación original no debe ser alterada en absoluto, su calidad y funcionamiento luego de la remoción ya debe estar asegurado con antelación.

Si se desea ejecutar la aplicación con funcionalidades volátiles, se deben definir las funcionalidades a incorporar y efectuar el deplyment de la aplicación original decorada. En cambio, si se desea ejecutar la aplicación original sin ninguna funcionalidad volátil añadida, simplemente basta con correr la aplicación original.

Tal como se discute en [10] algunos puntos que el framework propuesto debe cubrir son:

- Inyección de variables y relaciones, y mapeo de los mismos
- Inyección, modificación e intercepción de métodos
- Modificación de las vistas y archivos de configuración
- Mecanismo para hacer weaving entre las funcionalidades volátiles y la aplicación original
- Mecanismo para programar la habilitación y deshabilitación de las funcionalidades volátiles en tiempo de ejecución
- Mecanismo para correr la aplicación con varias funcionalidades volátiles a la vez

- Desacoplamiento de las funcionalidades volátiles en nuevos proyectos (el proyecto base original no debería ser alterado)

Otra característica interesante que brinda este enfoque es la posibilidad de programar la activación y la desactivación de las funcionalidades volátiles incorporadas. Mediante la definición de eventos temporales, eventos de negocio y/o combinaciones de ambos, pueden definirse los momentos para los cuales las distintas funcionalidades volátiles estarán o no disponibles. Estos cambios serán reflejados en tiempo de ejecución, proporcionándose así una implementación considerablemente dinámica. El hecho de poder configurar con antelación la activación y desactivación de funcionalidades volátiles resulta muy útil, dado que los cambios correspondientes serán reflejados en tiempo de ejecución, sin necesidad de volver a compilar y correr la aplicación.

La utilización del marco de trabajo propuesto podría resultar muy útil, ya que facilitaría, mejoraría y automatizaría la incorporación (y posterior remoción) de funcionalidades volátiles. Sin lugar a dudas, esto encajaría y podría ser aplicado perfectamente para el ejemplo que se ha dado para el lanzamiento de descuentos en aplicaciones web de empresas dedicadas a la comercialización de productos y servicios.

2.3 Estructura de trabajo

Anteriormente se dio una introducción al problema de la aparición de funcionalidades volátiles en aplicaciones web, también se mostró cuáles son las motivaciones que llevaron a su caso de estudio y posteriormente se analizará cómo puede enfrentarse dicho problema.

El trabajo desarrollado se focaliza en la utilización de un marco de trabajo que optimice la manipulación de las funcionalidades volátiles durante el ciclo de vida de las aplicaciones web.

El resto de la tesis está estructurada de la siguiente manera.

En el Capítulo 3, se describirán conceptos básicos con los que el lector deberá interiorizarse para lograr la comprensión de la tesis en general.

En el Capítulo 4, se discutirán trabajos relacionados con la investigación realizada.

En el Capítulo 5, se presentará el marco de trabajo “VF Framework” desarrollado, describiendo sus objetivos, sus características principales, su funcionamiento y la motivación para su creación.

Además, se describirán aspectos relacionados a las etapas por las que pasa una funcionalidad volátil durante su ciclo de vida, es decir desde que es introducida hasta que es eliminada de una aplicación web. Se mostrará qué ocurre tanto implementando funcionalidades volátiles de manera ad-hoc, como haciéndolo como VF Framework lo establece.

Otro tema importante que englobará este capítulo, será los requerimientos necesarios con los que un lenguaje de programación debe contar para poder añadir funcionalidades volátiles haciendo uso de este Framework. Para ello, se brindarán y analizarán las distintas alternativas existentes para cubrir cada uno de los puntos más importantes que comprende.

El Capítulo 6 se centrará en la implementación del VF Framework para el lenguaje de programación Java. Se detallarán las tecnologías utilizadas y las características técnicas necesarias para su implementación.

A continuación, se presentará el prototipo desarrollado para este lenguaje y finalmente se proporcionará una guía con la especificación de los pasos a seguir para incorporar funcionalidades volátiles haciendo uso de este marco de trabajo.

El Capítulo 7 se centrará en la implementación del VF Framework para el lenguaje de programación Smalltalk. Se detallarán las tecnologías utilizadas y las características técnicas necesarias para su implementación.

A continuación, se presentará el prototipo desarrollado para este lenguaje y finalmente se proporcionará una guía con la especificación de los pasos a seguir para incorporar funcionalidades volátiles haciendo uso de este marco de trabajo.

En el Capítulo 8, se procederá a comparar las implementaciones de los prototipos de VF Framework desarrollados para los lenguajes de programación Java y Smalltalk. Ambos prototipos serán contrastados indicando de manera resumida qué cuestiones fueron cubiertas y qué estrategias o tecnologías fueron utilizadas.

Además, el capítulo será cerrado con una pequeña conclusión obtenida luego de la implementación de estos dos prototipos.

En el Capítulo 9, se detallará el experimento realizado con el objetivo de obtener feed-back acerca del marco de trabajo presentado. Posteriormente, se analizarán y evaluarán los resultados obtenidos, y por último se brindarán las conclusiones al respecto.

En el Capítulo 10, se definirán los trabajos futuros a efectuar con el fin de continuar y avanzar con la investigación realizada.

Finalmente se concluirá con el Capítulo 11, en donde se darán las conclusiones finales acerca del trabajo de tesis.

3 Conceptos Básicos

En este Capítulo se describirán conceptos básicos con los que el lector deberá interiorizarse para lograr la comprensión de la tesis en general.

3.1 Aplicación Web

Tal cual se establece en [11], se denomina aplicación web a cualquier aplicación que utilice un navegador como cliente. La misma puede tener distintos niveles de complejidad, variando desde el más simple hasta el más complejo.

El término "cliente" se emplea dentro del ambiente cliente-servidor para referirse al programa que utiliza la persona para correr la aplicación. En dicho ambiente varias computadoras comparten información, tal como los datos de una base de datos. El "cliente" es la aplicación utilizada para cargar la información, y el "servidor" es la aplicación utilizada para procesar y almacenar dicha información.

Uno de los principales beneficios de las aplicaciones web es que liberan al desarrollador de la responsabilidad de construir un cliente para un tipo específico de computadora o de sistema operativo.

Las aplicaciones web habitualmente utilizan una combinación de scripts del lado del servidor (ASP, PHP, etc.) y scripts del lado del cliente (HTML, JavaScript, etc.) para su desarrollo. Los scripts del lado del cliente se ocupan de la presentación de la información, mientras que los scripts del lado del servidor se ocupan de todas las tareas más complejas, como el procesamiento, el almacenamiento y la recuperación de la información. Un claro ejemplo de ello son los WebMails que empresas como Google, Yahoo y MSN ofrecen a sus clientes.

Las aplicaciones web han existido antes de que la web ganara popularidad. Por ejemplo, en 1987 Larry Wall desarrolló Perl, un popular lenguaje de escritura del lado del servidor. Esto fue siete años antes de que Internet se volviera realmente popular fuera de los círculos académicos y tecnológicos.

En la actualidad, diariamente millones de personas emplean aplicaciones web para realizar todo tipo de tareas tales como leer el periódico, comprar productos y pagar servicios. La posibilidad de realizar esto en la web, sin lugar a dudas, facilita la vida de las personas. Por esta razón, cada vez son más las tareas que pueden realizarse vía web y por ende la necesidad de construir y mejorar más aplicaciones web.

3.2 OOHDM

En esta sección introduciremos brevemente la metodología OOHDM (Object Oriented Hypermedia Design Method) pudiendo contar con información más específica en [14,15].

OOHDM considera al desarrollo de una aplicación hipermedia como un proceso compuesto por cuatro actividades: la definición del esquema conceptual del dominio de la aplicación, el diseño del esquema navegacional, la especificación de la interfaz del usuario y la implementación.

La definición del esquema conceptual del dominio de la aplicación se realiza utilizando los principios de diseño de la programación orientada a objetos y otras primitivas, tales como la definición de atributos y de subsistemas.

Una de las características principales de las aplicaciones web es la noción de navegación. En OOHDM, se considera a una aplicación web como una vista

navegacional del modelo conceptual. El diseño de esta vista es lo que se obtiene en la etapa de diseño del esquema navegacional teniendo en cuenta los diferentes tipos de usuarios que tendrá la aplicación. El esquema navegacional puede ser recorrido mediante diferentes contextos. Un contexto navegacional es un conjunto de nodos relacionados que poseen una determinada estrategia de recorrido.

En la especificación de la interfaz del usuario se definen los objetos que serán vistos por los usuarios, la representación que tendrán los objetos navegacionales, los objetos que permitirán la navegación, la sincronización entre los objetos que brindan funcionalidad multimedia y las transformaciones que sucederán en la interfaz mientras el usuario navega.

Finalmente, en la etapa de implementación, se realiza el mapeo de los objetos del esquema navegacional y los de la interfaz a un lenguaje de programación. Para que en el mapeo no perdamos las propiedades que brinda la programación orientada a objetos que posee nuestro modelo, necesitamos un lenguaje con tales características.

3.3 Ciclo de vida del desarrollo de software

Como lo dice en [22], existen varios enfoques definidos y diseñados para el desarrollo de software, que son utilizados durante el proceso de desarrollo de software. Dichos enfoques también son denominados "Modelos para el Proceso de Desarrollo de Software"(ej. modelo en Cascada, modelo incremental, modelo V, modelo iterativo, etc.). Cada modelo de proceso sigue un ciclo de vida particular para así asegurar el éxito en el desarrollo del proceso de software.

Los modelos de ciclo de vida describen fases del ciclo de software y el orden en el cual dichas fases son ejecutadas. Cada fase produce salidas requeridas por la próxima fase en el ciclo de vida. Los requerimientos son traducidos al diseño. El código es producido de acuerdo al diseño, lo que se denomina fase de desarrollo. Luego de la codificación y del desarrollo, el testing verifica la salida de la fase de implementación en relación a los requerimientos.

Hay seis fases en el ciclo de vida de todo modelo de desarrollo de software:

1. **Análisis y elicitación de requerimientos**
2. **Diseño**
3. **Implementación o codificación**
4. **Testing**
5. **Deployment**
6. **Mantenimiento**

1) **Análisis y elicitación de requerimientos:** Los requerimientos de negocio son recogidos en esta fase. La misma es el foco principal de los project managers y de los accionistas. Las reuniones con los managers, accionistas y usuarios son realizadas con el objeto de determinar requerimientos tales como: ¿Quién va a utilizar el sistema? ¿Cómo utilizarán el sistema? ¿Qué datos deberían ser ingresados en el sistema? ¿Qué datos deberían ser arrojados por el sistema? Estas son preguntas generales que son respondidas durante una fase de elicitación de requerimientos. Luego de la misma, estos requerimientos son analizados para determinar su validez; asimismo también se estudia la

posibilidad de incorporar los requerimientos en el sistema a ser desarrollado. Finalmente, se crea un documento de especificación de requerimientos, el cual señala las pautas para la próxima fase del modelo.

2) **Diseño:** En esta fase el sistema y el diseño de software son preparados de acuerdo a las especificaciones de los requerimientos que fueron estudiados durante la primera fase. El diseño de sistema ayuda a especificar los requerimientos del hardware y del sistema, y también ayuda a definir a la arquitectura del sistema en su conjunto.
   Las especificaciones del diseño de sistema sirven como entrada para la próxima fase del modelo.
3) **Implementación / Codificación:** Al recibir documentos de diseño de sistema, el trabajo es dividido en módulos/unidades y comienza la codificación real. El código es producido en esta fase, motivo por el cual se convierte en el foco principal para el desarrollador.
4) **Testing:** Una vez que el código ha sido desarrollado, el mismo se compara con los requerimientos con el objeto de asegurar que el producto realmente sea capaz de resolver las necesidades identificadas y elicitadas durante la fase de requerimientos. Durante dicha fase se llevan a cabo el testing de unidad, el testing de integración, el testing del sistema, y el testing de aceptación.
5) **Deployment:** Luego de un testing exitoso, se realiza la entrega y el deployment del producto para el uso del consumidor.
6) **Mantenimiento:** Una vez que el cliente comienza a hacer uso del sistema desarrollado es cuando surgen los problemas reales que requieren una solución inmediata. Este proceso durante el cual se mantiene y se da soporte para el producto desarrollado se denomina mantenimiento.

3.3.1 Mantenimiento

Las funcionalidades volátiles tienen gran impacto en esta etapa del ciclo de vida del desarrollo de software. Por este motivo en esta sección se incluirán más especificaciones con respecto a dicha etapa. La información pertinente fue obtenida de [23].

Una vez que el producto ha sido lanzado, la fase de mantenimiento mantiene el software actualizado con los cambios del ambiente, así como también con los requerimientos cambiantes del usuario.

El mantenimiento consta de cuatro partes. El mantenimiento correctivo se ocupa de corregir los bugs que aparecen en el código. El mantenimiento de adaptación se encarga de adaptar el software a los nuevos ambientes. El mantenimiento de perfeccionamiento trata la actualización del software en relación a los cambios en los requerimientos del usuario. Finalmente, el mantenimiento preventivo se encarga de actualizar la documentación y de volver al software más mantenible. Todos los cambios al sistema pueden ser caracterizados por estos cuatro tipos de mantenimiento. El mantenimiento correctivo es el "mantenimiento tradicional", mientras que los otros tipos son considerados "evolución de software".

Resumen de procesos

Los modelos de ciclo de vida tradicionales no toman en cuenta la naturaleza evolutiva de los sistemas de software; por consiguiente, se requieren diferentes modelos para lograr mantenibilidad. La comprensión del programa resulta crucial dado que se invierte más de la mitad del tiempo y del esfuerzo en efectuar cambios. La mejora del

rendimiento en los trabajos de mantenimiento conducirá a una mayor productividad y a una evolución exitosa de productos de software. La reutilización de software también aumenta la productividad y mejora la mantenibilidad y la calidad del sistema de software empleando los componentes de software existentes.

Tareas

Las tareas de mantenimiento puede ser agrupadas en cinco categorías:

1. Análisis
2. Diseño
3. Implementación
4. Testing
5. Deployment

1) **Análisis:** Las tareas de análisis consisten en análisis de impacto, análisis de costo-beneficio y aislamiento. El análisis de impacto y el análisis de costo-beneficio consisten en analizar diferentes alternativas de implementación y comparar su efecto sobre el tiempo, costo y operación. El aislamiento se refiere al tiempo pasado en tratar de comprender el problema o las mejoras al sistema propuestas.
2) **Diseño:** El diseño consiste en rediseñar el sistema basándose en la comprensión de los cambios necesarios. También supone documentación semiformal, como la publicación de documentos de revisión.
3) **Implementación:** La implementación comprende el testing de código y unidad. El mismo se refiere al tiempo invertido en codificar y testear los cambios. También comprende documentación semiformal, tal como el plan de testeo de modificación de software. El testing de unidad es llevado a cabo por el encargado de realizar el mantenimiento que ha efectuado los cambios. Usualmente el testing de unidad se realiza localmente en la estación de trabajo del usuario.
4) **Testing:** El testing comprende integración, aceptación y regresión. El testing de integración se refiere al tiempo invertido en la integración de los componentes, mientras que el testing de aceptación implica verificar que el sistema modificado se adhiere a los requerimientos del usuario. El testing de aceptación es realizado por los usuarios finales con el fin de asegurar que los cambios deseados hayan sido implementados exitosamente. El testing de regresión se refiere al tiempo invertido en asegurar que los cambios no hayan afectado la funcionalidad de las otras partes del software.
5) **Deployment:** El deployment consiste en volver a efectuar la entrega y la ejecución del producto desarrollado para el uso del consumidor. Esta tarea se da luego de haber concluido satisfactoriamente las etapas anteriores.

3.4 Programación Orientada a Aspectos (AOP)

La programación orientada a Aspectos tiene un papel fundamental para esta tesis. Es por ello que en esta sección será citado material obtenido de [24].

Separación de concerns

La separación de concerns es un principio importante de la ingeniería de software. Se refiere a la habilidad de identificar, encapsular y manipular aquellas partes de software que son relevantes para un concern particular (concepto, objetivo, propósito, etc.). Los concerns pueden variar desde notaciones de alto nivel, como seguridad y calidad de servicios, a notaciones de bajo nivel, como buffering, caching y logging. Asimismo, pueden ser funcionales, como la lógica de negocios, o no funcionales, como la sincronización. Un sistema típico consta de varios concerns. En la forma más simple se encuentran los concerns centrales, es decir los componentes naturales del software. Salvo por estos concerns centrales, hay concerns del nivel del sistema, como seguridad, logging, autenticación, persistencia, etc., que tienden a afectar otros varios concerns. Por ejemplo, si se va a implementar una característica del logging en una aplicación, es factible que todos los módulos subyacentes tengan un código para logging, volviendo a los módulos subyacentes menos especializados y dificultando el poder predecir qué efectos tendrán los cambios en el código para el logging. A pesar de que la programación orientada a objetos (OOP) ofrece cierta habilidad para separar los concerns, aún conserva dificultad para localizar concerns que no encajan naturalmente en un solo módulo de programa, o inclusive varios módulos de programa estrechamente relacionados.

Cross-cutting concerns

Los problemas discutidos anteriormente son denominados cross-cutting concerns, dado que atraviesan otros varios módulos en el sistema. Los crosscutting concerns son conductas que abarcan módulos de implementación múltiples y a menudo no relacionados. Además, los crosscutting concerns no pueden ser separados eficientemente unos de otros. Son ejemplos típicos de crosscutting concerns:

- Seguridad (autorización y auditoría)
- Logging y debugging
- Sincronización
- Persistencia

Paradigma AOP

La programación Orientada a Aspectos (AOP) es un paradigma de programación que se esfuerza por ayudar al desarrollador a separar los concerns. El mismo apunta a superar los problemas con los crosscutting concerns descriptos anteriormente, y provee mecanismos de lenguaje que capturan explícitamente la estructura crosscutting. Esto hace posible que los crosscutting concerns sean programados de una forma modular, y se obtengan los beneficios de una modularidad mejorada: un código más simple que es más fácil de desarrollar y mantener, y que tiene un mayor potencial de reutilización. Esto se logra mejorando los aspectos de uso del código de modularización.

Lo que OOP ha hecho para el encapsulamiento y la herencia de objetos, es lo que AOP hace para los crosscutting concerns. Un aspecto es, por definición, unidades modulares que cruzan la estructura de otras unidades. Un aspecto es similar a una clase en cuanto a que tiene un tipo, puede extender clases y otros aspectos, puede ser abstracto o concreto y tener campos, métodos, y tipos como miembros. Encapsula conductas que afectan clases múltiples y las convierte en módulos reusables.

Un dato importante es que AOP no realiza crosscutting concerns a ninguna otra cosa salvo un crosscutting concern. AOP convertirá una implementación enredada y dispersa de un crosscutting concern en una implementación modularizada de un

crosscutting concern. Cuando se aplica AOP, el concern seguirá siendo crosscutting, pero la estructura será más clara.

En los lenguajes AOP, los aspectos utilizan tres elementos esenciales:

- **Joinpoints:** son puntos bien definidos en la ejecución de un programa como llamadas a métodos, accesos a campos, chequeos condicionales, comienzos de loops, asignaciones y construcciones de objetos.
- **Pointcuts:** son construcciones de programas para designar joinpoints y reunir un contexto específico en esos puntos. Los criterios pueden ser nombres de función explícitos o bien nombres de función especificados por wildcards.
- **Advices:** son conductas especificadas para ejecutarse ante el cumplimiento de ciertos pointcuts. En AspectJ por ejemplo, hay tres advices diferentes; antes del advice, después del advice y alrededor del advice.

3.5 Data Mapper

Las aplicaciones web requieren el almacenamiento de datos en bases de datos. Allí se guarda información vital para el funcionamiento y para el negocio para el cual las aplicaciones Web han sido creadas. Para lograr este objetivo, suele hacerse uso de Data Mappers. Los Data Mappers son justamente los encargados de mapear esa información en las bases de datos.

Los lenguajes de programación orientados a objetos (que son en los que se enfoca esta tesis), utilizan en su mayoría Object Relational Mappers (ORM) como Data Mappers. Un ORM es una capa de software que ayuda a mapear objetos en tablas de bases de datos relacionales. Algunos ORMs manejan más aspectos que otros, pero el objetivo principal de todos ellos es el de quitar algo del peso de la capa de datos de los hombros del desarrollador.

Tal cual sostiene Martin Fowler en [29], los objetos y las bases de datos relacionales poseen diferentes mecanismos para estructurar los datos. Varias partes de un objeto, como las colecciones y la herencia, no están presentes en las bases de datos relacionales. Cuando se construye un modelo de objeto con mucha lógica de negocios, es valioso utilizar estos mecanismos para organizar mejor los datos y el comportamiento que lo acompaña. Esto conduce a varios esquemas, lo cual a su vez trae aparejado que el esquema del objeto y el esquema relacional no coincidan.

Aún así, resulta necesaria la transferencia de datos entre ambos esquemas, y esta transferencia se vuelve una complejidad en sí misma. Si los objetos en memoria saben acerca de la estructura de la base de datos relacional, los cambios en uno tienden a repercutir en el otro.

El Data Mapper es una capa de software que separa los objetos en memoria de la base de datos. Su responsabilidad es transferir datos entre ambos y también aislarlos el uno del otro. Con Data Mapper los objetos en memoria no necesitan ni siquiera saber que hay una base de datos presente; no necesitan código de interfaz SQL, ni conocer el esquema de la base de datos. Asimismo, el esquema de la base de datos siempre desconoce los objetos que lo utilizan. Dado que es una forma de mapeador, el propio Data Mapper es desconocido para la capa de dominio.

En la Ilustración 2 puede observarse un ejemplo del Data Mapper “Person Mapper”, encargado de mapear la clase “Person” a una base de datos.

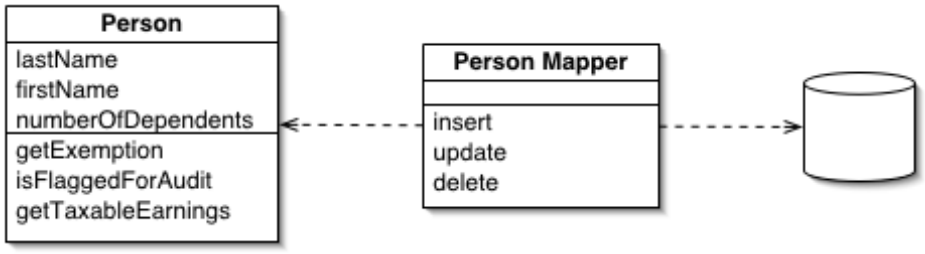


Ilustración 2: Data Mapper

3.6 Expresiones regulares

Las expresiones regulares han sido utilizadas para el desarrollo de la tesis y por ende las mismas serán introducidas brevemente en esta sección a partir de [25].

Una expresión regular, también denominada regex, es una secuencia de caracteres que forma un patrón de búsqueda. Cuando se buscan datos en un texto, se puede utilizar este patrón de búsqueda para describir lo que se está buscando. Cabe destacar que una expresión regular puede ser tanto un solo carácter, como también un patrón más complicado.

Las expresiones regulares pueden ser utilizadas para realizar todo tipo de búsqueda de texto y operaciones de reemplazo de texto. Para este segundo caso, es necesario enviar como parámetros un argumento de búsqueda y un argumento de reemplazo.

Ejemplo en JavaScript:

```
function myFunction() {
        var str = "Mr Blue has a blue house and a blue car."

  //Agrega el texto ", a blue hat" antes de la palabra "and"
        str = str.replace(/\sand/, ", a blue hat and");

  //Cambia las palabras "blue" (sin prestarle atención a las mayúsculas) por "red"
        str = str.replace(/blue/gi, "red");

  //Pone las palabras "house", "hat" y "car" en mayúscula

  str = str.replace(/house|hat|car/g, function myFunction(x){return x.toUpperCase();});
        //Cambie el caracter "." por el caracter "!".
        str = str.replace(/\./, "!");
        alert(str);
    }
```

Cuando esta función sea llamada, se visualizará un diálogo con el mensaje: "Mr red has a red HOUSE, a red HAT and a red CAR!".

3.7 XSLT

XSLT (Transformaciones XSL) es un lenguaje de programación declarativo que permite generar nuevos documentos a partir de la transformación de documentos XML. Este estándar de la organización W3C ha resultado útil para el desarrollo de la tesis, razón por la cual será explicado brevemente. Para ello se ha obtenido información de [27] y de [28].

Los documentos XSLT son empleados para transformar documentos XML en otros documentos XML, o inclusive en otros tipos de documentos reconocibles por los navegadores como HTML y XHTML. Normalmente, XSLT realiza esto transformando cada elemento XML en un elemento (X)HTML.

Con XSLT se pueden generar archivos de salida en los cuales se remueven o añaden elementos y atributos con respecto a los archivos de entrada. También permite realizar otros tipos de tareas como ordenar y reorganizar elementos, efectuar tests y tomar decisiones acerca de qué elementos mostrar, entre otros.

Las hojas de estilo XSLT realizan la transformación del documento utilizando una o varias reglas de plantilla. Estas reglas de plantilla unidas al documento fuente a transformar alimentan un procesador de XSLT, encargado de realizar las transformaciones deseadas poniendo el resultado en un nuevo archivo de salida.

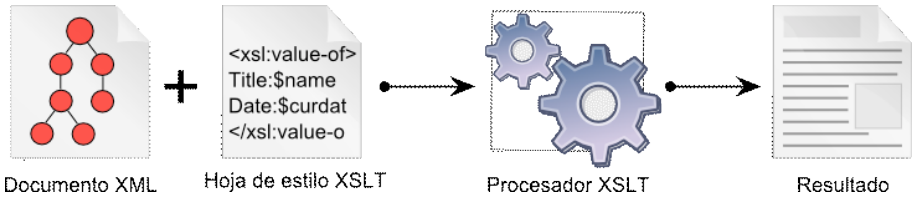


Ilustración 3: Proceso de XSLT

En la Ilustración 3, se visualiza dicho proceso.

- El documento XML es el documento inicial a partir del cual se va a generar el resultado.
- La hoja de estilo XSLT es el documento que contiene el código fuente del programa, es decir, las reglas de transformación que se van a aplicar al documento inicial.
- El procesador XSLT es el programa de ordenador que aplica al documento inicial las reglas de transformación incluidas en la hoja de estilo XSLT y genera el documento final.
- El resultado de la ejecución del programa es un nuevo documento (que puede ser un documento XML o no).

A continuación se muestra un ejemplo de una transformación XSLT.

Documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
	<cd>
		<title>Empire Burlesque</title>
		<artist>Bob Dylan</artist>
		<country>USA</country>
		<company>Columbia</company>
```

```
        <price>10.90</price>
        <year>1985</year>
    </cd>
    <cd>
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <country>UK</country>
        <company>CBS Records</company>
        <price>9.90</price>
        <year>1988</year>
    </cd>
    <cd>
        <title>Greatest Hits</title>
        <artist>Dolly Parton</artist>
        <country>USA</country>
        <company>RCA</company>
        <price>9.90</price>
        <year>1982</year>
    </cd>
    <cd>
        <title>Still got the blues</title>
        <artist>Gary Moore</artist>
        <country>UK</country>
        <company>Virgin records</company>
        <price>10.20</price>
        <year>1990</year>
    </cd>
</catalog>
```

Hoja de estilo XSLT:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <xsl:if test="price>10">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
        <td><xsl:value-of select="price"/></td>
```

```
                </tr>
            </xsl:if>
            </xsl:for-each>
            </table>
          </body>
          </html>
        </xsl:template>
        </xsl:stylesheet>
```

Resultado HTML:

```
        <html>
          <head>
          </head>
          <body>
            <h2>My CD Collection</h2>
            <table border="1">
              <tbody>
                <tr bgcolor="#9acd32">
                  <th>Title</th>
                  <th>Artist</th>
                  <th>Price</th>
                </tr>
                <tr>
                  <td>Empire Burlesque</td>
                  <td>Bob Dylan</td>
                  <td>10.90</td>
                </tr>
                <tr>
                  <td>Still got the blues</td>
                  <td>Gary Moore</td>
                  <td>10.20</td>
                </tr>
              </tbody>
            </table>
          </body>
        </html>
```

Visualización del resultado HTML:

My CD Collection

Title	Artist	Price
Empire Burlesque	Bob Dylan	10.90
Still got the blues	Gary Moore	10.20

4 Trabajos relacionados

En este Capítulo, se discutirán trabajos relacionados con la investigación realizada.

4.1 Feature Oriented Software Development (FOSD)

Como trabajo relacionado puede hacerse mención a [26], en donde se hace referencia a FOSD.

El FOSD es un paradigma para la construcción, personalización y síntesis de sistemas de software de gran escala. El concepto de "feature" se encuentra en el corazón del FOSD. Una "feature" es una unidad de funcionalidad de un sistema de software que satisface un requerimiento, representa una decisión de diseño y provee una opción de configuración potencial. La idea básica del FOSD es descomponer un sistema de software en relación a las "features" que provee. El objetivo de la descomposición es construir un software bien estructurado, que pueda ser personalizado de acuerdo a las necesidades del usuario y del escenario de aplicación. Típicamente, varios sistemas de software diferentes pueden ser generados de un conjunto de "features" que comparten "features" comunes y difieren en otras. El conjunto de sistemas de software generado de un conjunto de "features" también es denominado una línea de producto de software. El FOSD tiene por objetivo esencialmente tres propiedades: estructura, reutilización y variación. Los desarrolladores emplean el concepto de "feature" para estructurar el diseño y el código de un sistema de software. Las "features" son las unidades primarias de reutilización en FOSD, y las variantes de un sistema de software varían en las "features" que proveen.

A diferencia del enfoque presentado en esta tesis, el FOSD apunta a la construcción de software a partir de la composición de funcionalidades. En lugar de utilizar aspectos para integrar ciertas funcionalidades, utiliza la composición algebraica para la integración de todas las funcionalidades del sistema. Si bien ambos enfoques poseen similitudes como el desacoplamiento de funcionalidades, el FOSD no considera el hecho de programar eventos para la activación y/o desactivación de funcionalidades en tiempo de ejecución.

4.2 Aspectos dinámicos

En [19] se hace referencia al uso de aspectos dinámicos para un lenguaje puramente orientado a objetos y de tipado dinámico como lo es Smalltalk. Para ello se presenta "Dynamic Aspects", una implementación ligera de AOP para Smalltalk, que se beneficia de las herramientas avanzadas de "reflection". La "reflection" de nivel de sub método permite seleccionar sentencias únicas dentro de los métodos. La anotación de dichas sentencias permite la adición extrínseca del comportamiento en cualquier sector del código. Finalmente, la idea de la "reflection" parcial del comportamiento le otorga gran flexibilidad al sistema. Todas esas herramientas son el corazón sobre el cual está construido "Dynamic Aspects". Más allá de la implementación básica de AOP, "Dynamic Aspects" está relacionado con el campo de la programación orientada al contexto, y muestra cómo los contextos pueden ser utilizados en aspectos. Asimismo, la idea de control de flujo es generalizada, y el flujo genérico es implementado con contextos.

Esta implementación AOP permite la activación y desactivación de aspectos en tiempo de ejecución. También se muestra como a partir de aspectos dinámicos es posible definir pointcuts y advices asociados, capaces de modificar las distintas capas de una aplicación web. Sin embargo, aquí no se hace mención alguna acerca de la inyección de métodos y variables, ni acerca de la programación de activaciones y desactivaciones a partir de eventos.

4.3 Motor de reglas

Como establece Martin Fowler en [30], un motor de reglas es una alternativa de modelo computacional. En lugar del habitual modelo imperativo con comandos en secuencia con condicionales y loops, provee una lista de reglas de producción. Cada regla consta de una condición y una acción a ejecutar. Simplísiticamente se lo puede ver como un montón de sentencias if-then.

Las reglas pueden ser escritas en cualquier orden, siendo el motor quien finalmente decide cuándo evaluarlas, empleando cualquier orden que le resulte lógico. El sistema repasa todas las reglas, escoge aquellas par las cuales la condición es verdadera, y luego evalúa las acciones correspondientes.

Este modelo encaja y sirve para representar una amplia variedad de problemas. A continuación se muestra lo que podría ser un típico caso para la utilización de un set de reglas:

```
if (car.owner.hasCellPhone) then premium += 100;
if (car.model.theftRating > 4) then premium += 200;
if (car.owner.livesInDodgyArea && car.model.theftRating > 2) then premium += 300;
```

En este ejemplo, se calcula el monto a pagar por el seguro de un auto a partir de la definición de un conjunto de reglas.

Un motor de reglas es una herramienta que facilita la programación empleando este modelo computacional. Puede ser un ambiente de desarrollo completo, o un Framework que pueda trabajar con una plataforma tradicional. El modelo computacional de reglas de producción resulta más adecuado para sólo un subconjunto de problemas computacionales, por lo tanto los motores de reglas resultan mejor si están embebidos en sistemas más grandes.

Resulta muy fácil construir un motor de reglas simple; todo lo que hace falta es crear un montón de objetos con condiciones y acciones, almacenarlos en una colección, y correrlos para evaluar las condiciones y ejecutar las acciones. Sin embargo, cuando la gente hace mención a un "motor de reglas", generalmente se refiere al producto construido específicamente para ayudar a crear y correr un motor de reglas. Las técnicas para especificar las reglas pueden variar desde una API para describir reglas como objetos de Java, un archivo DSL para expresar reglas, o un GUI que permita ingresar reglas. Los motores de ejecución más eficientes ayudan a evaluar rápidamente las condiciones en cientos de reglas utilizando algoritmos especializados (tales como el algoritmo Rete).

Los motores de reglas son una herramienta muy útil y flexible, que permite que personas relacionadas al negocio puedan definir reglas por su cuenta, sin contar con grandes conocimientos informáticos, ni requerir la ayuda de programadores en el proceso. No obstante, las reglas definidas apuntan principalmente a consideraciones y

cambios relacionados con la lógica de negocio de la aplicación. Su objetivo principal no incluye la gestión de otro tipo de cambios tales como la modificación del modelo de datos, la modificación de las capa de presentación y demás cambios que podrían ser requeridos ante la aparición de funcionalidades volátiles. Por esta razón, no son suficiente para ocuparse de ellas.

Cabe destacar igualmente que, como será detallado en el Capítulo 5, los motores de reglas pueden formar parte de la solución desarrollada para cumplir dicho objetivo.

4.4 Framework J2EE

Los Frameworks J2EE son marcos de trabajo que sirven para simplificar y proveer facilidades a la hora de desarrollar aplicaciones web con el lenguaje de programación Java. El hecho de utilizar un Framework de este tipo indudablemente brinda grandes ventajas que proveen un apoyo integral de infraestructura para el desarrollo de aplicaciones. De este modo, los desarrolladores pueden enfocarse más asiduamente en la construcción de su aplicación.

Spring Framework es un claro ejemplo de Framework J2EE utilizado masivamente. El mismo representa una solución ligera, completa y potente para la construcción de aplicaciones. Además, está diseñado para ser no intrusivo, lo cual significa que el código de lógica de dominio generalmente no depende del Framework en sí mismo.

Spring Framework consiste en features organizados en aproximadamente 20 módulos. Tal como se muestra en el diagrama de la Ilustración 4, estos módulos están agrupados en las categorías: Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Aspects, Instrumentation, Messaging y Test. Cabe destacar que como Spring Framework es modular, este permite hacer uso sólo de aquellas partes que sean requeridas. Puede obtener más información al respecto en [5].

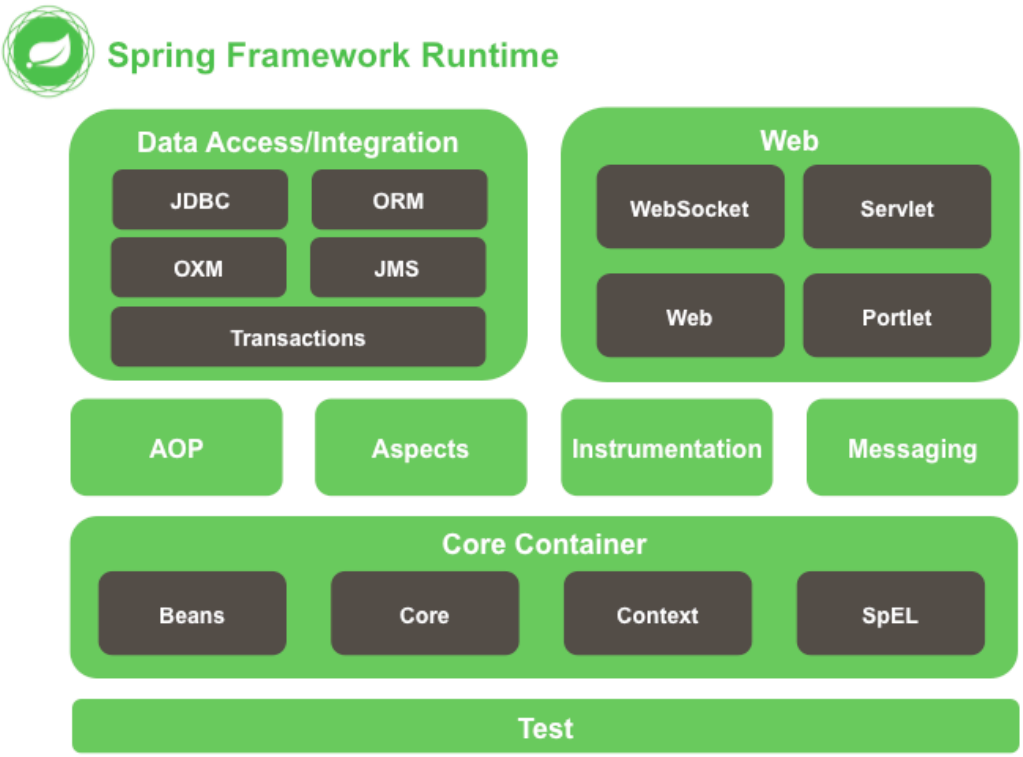


Ilustración 4: Resumen del Framework Spring MVC

Como ha sido explicado, este tipo de Frameworks resultan muy útiles e inclusive han sido utilizados para el desarrollo de un prototipo que será descripto más adelante en el Capítulo 6. Sin embargo, estos Frameworks por sí solos no alcanzan, ni cuentan con las herramientas necesarias para cubrir la situación presentada de la gestión de las funcionalidades volátiles. Esto se debe básicamente a que las funcionalidades volátiles son un tema que no está siquiera puesto en consideración.

5 VF Framework

5.1 Descripción

VF Framework tiene como objetivo la definición de un marco de trabajo conceptual y tecnológico para la implementación, incorporación, activación y desactivación de funcionalidades volátiles en aplicaciones web de manera desacoplada.

La conservación de la independencia y la integridad de la aplicación base original resulta primordial para lograr este objetivo. Por esta razón, es necesario ubicar desacopladamente por un lado el código correspondiente a la aplicación original que permanecerá intacto, y por otro lado todo el código correspondiente a las funcionalidades volátiles que será utilizado para enriquecer a la aplicación original. Para ello, las funcionalidades volátiles a introducir deben ser capaces de modificar todas las capas pertenecientes a la aplicación original (es decir la capa de presentación, la capa de navegación y la capa de negocio).

En la Ilustración 5 puede visualizarse una representación gráfica del proceso recién descripto. En primer lugar, se encuentra la "Aplicación Base" compuesta por sus correspondientes capas. En segundo lugar, se encuentran las "Mejoras FV" compuestas por las modificaciones a efectuar sobre cada una de las capas de la "Aplicación Base". Finalmente, a partir de la integración entre las dos anteriores se obtiene como resultado la "Aplicación Decorada", la cual está compuesta por las capas de la "Aplicación Base" modificadas por las "Mejoras FV".

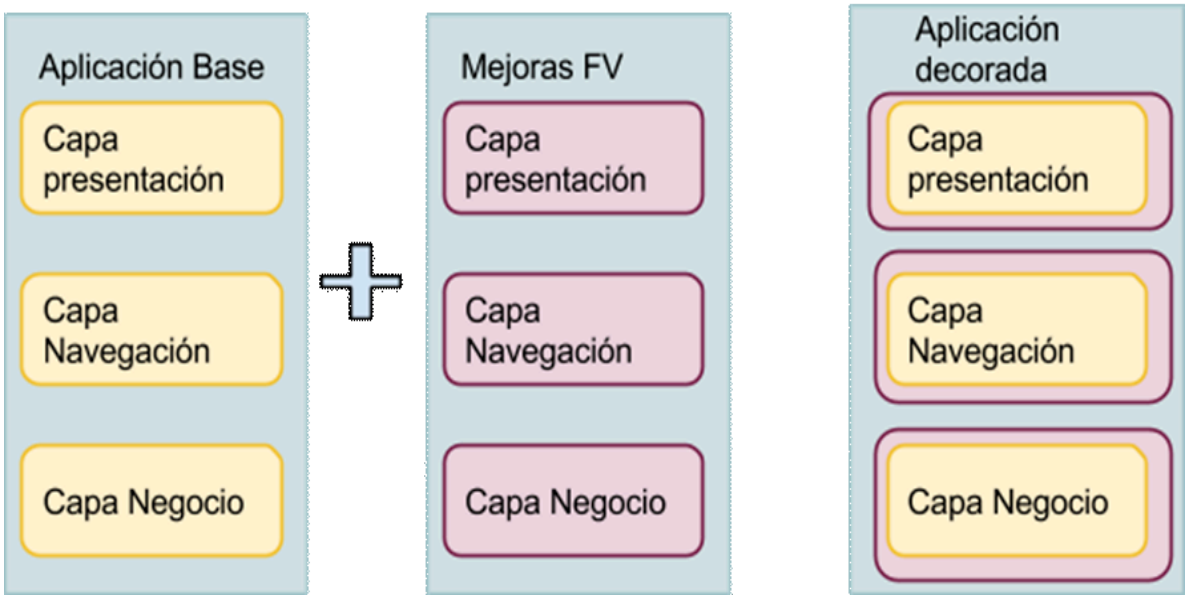


Ilustración 5: Aplicación decorada con FVs

Una ventaja importante que provee el marco de trabajo propuesto se da al optar por pasar de una versión de la aplicación con determinadas funcionalidades volátiles incorporadas a una versión con menos funcionalidades volátiles o inclusive a la versión original (sin ninguna funcionalidad volátil del todo). El proceso para llevar a cabo este tipo de remociones se ve muy simplificado. Esto es así dado que como la aplicación original no debe ser alterada en absoluto, su calidad y funcionamiento luego de la remoción ya debe estar asegurado con antelación.

Si se desea ejecutar la aplicación con funcionalidades volátiles, se deben definir las funcionalidades a incorporar y efectuar el deployment de la aplicación original decorada. En cambio, si se desea ejecutar la aplicación original sin ninguna funcionalidad volátil añadida, simplemente basta con correr la aplicación original.

Otra característica interesante que brinda este enfoque es la posibilidad de programar la activación y la desactivación de las funcionalidades volátiles incorporadas. Mediante la definición de eventos temporales, eventos de negocio y/o combinaciones de ambos, pueden definirse los momentos para los cuales las distintas funcionalidades volátiles estarán o no disponibles. Estos cambios serán reflejados en tiempo de ejecución, proporcionándose así una implementación considerablemente dinámica. El hecho de poder configurar con antelación la activación y desactivación de funcionalidades volátiles resulta muy útil, dado que los cambios correspondientes serán reflejados en tiempo de ejecución, sin necesidad de volver a compilar y correr la aplicación.

Teniendo en cuenta que la etapa de mantenimiento es la que mayor esfuerzo representa en el ciclo de vida del desarrollo de un proyecto, el hecho de poder optimizarla, significaría no sólo la obtención de mejores resultados en cuanto a calidad, sino también la reducción del esfuerzo de trabajo, del tiempo y, por ende, de los costos.

Nota: Cabe desatacar que si bien el VF Framework puede ser muy beneficioso, su utilización realmente debe justificarse. No tendría ningún sentido implementar estos conceptos para incorporar funcionalidades que deberían formar parte del núcleo de la aplicación y que no requerirían ser desactivadas en el futuro.

5.2 Ciclo de vida de las funcionalidades volátiles

En base a [10], en las siguientes subsecciones se describirán aspectos relacionados a las etapas por las que pasan las funcionalidades volátiles durante su ciclo de vida, es decir desde que son introducidas hasta que son eliminadas de una aplicación web. Inclusive se analizarán los casos de aquellas funcionalidades volátiles que reaparecen posteriormente a su remoción.

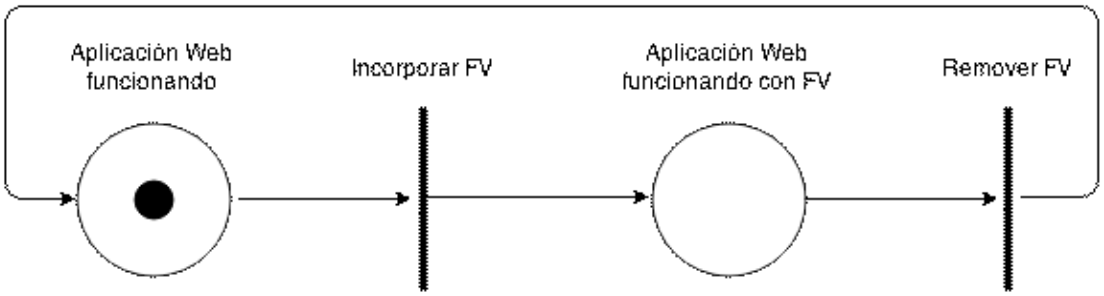


Ilustración 6: Red de Petri del ciclo de vida de una FV

En la Ilustración 6, se ha representado el ciclo de vida de una funcionalidad volátil de manera general y sintetizada. Para ello, se ha utilizado una Red de Petri que muestra cómo una aplicación web puede modificar su estado a partir de la introducción y la remoción de una funcionalidad volátil. Cuando una aplicación web se encuentra en funcionamiento, puede producirse la ocurrencia de un evento que conlleve la incorporación de una funcionalidad volátil. Cuando esto suceda, la aplicación pasará a ser ejecutada con la funcionalidad volátil en cuestión incorporada. Eventualmente, esta

funcionalidad volátil caducará y deberá ser removida. Entonces, se producirá un evento de remoción y la aplicación web será ejecutada en su estado original nuevamente. Este ciclo puede repetirse una múltiple cantidad de veces.

A continuación se mostrará de manera más detallada qué ocurre, tanto implementando funcionalidades volátiles de manera ad-hoc o tradicional, como haciéndolo como VF Framework lo establece.

5.2.1 Incorporación de funcionalidades volátiles

Las funcionalidades volátiles tienen su origen durante la etapa de mantenimiento del ciclo de vida del desarrollo de software. Una vez identificadas, para llevar a cabo su tratamiento se requiere un mantenimiento de perfeccionamiento que permita cumplir con las nuevas necesidades solicitadas.

Sin importar el tipo de implementación utilizada para la incorporación de las funcionalidades volátiles solicitadas, serán necesarias las siguientes etapas:

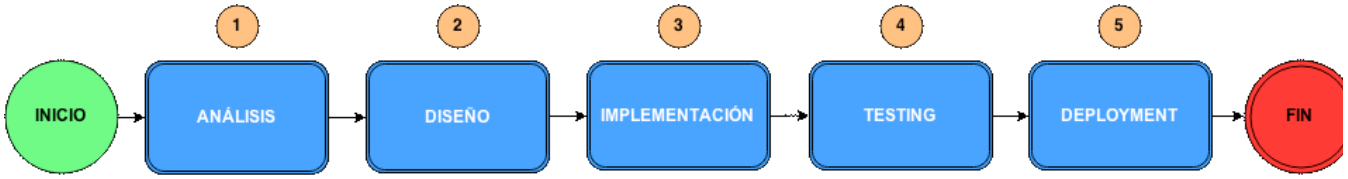


Ilustración 7: Etapas para la incorporación de una FV

1) **Análisis:** Se detecta y analiza la funcionalidad volátil a incorporar en cuanto a cuestiones tales como impacto, costo-beneficio, y aislamiento.
2) **Diseño:** Se rediseña el sistema basándose en la comprensión de los cambios necesarios a partir de la incorporación de la funcionalidad volátil.
3) **Implementación:** Se codifica y se prueban los cambios introducidos mediante tests de código y de unidad.
4) **Testing:** Se realizan principalmente pruebas de integración y regresión para asegurar el correcto funcionamiento de la aplicación. Esta etapa suele requerir la ejecución de múltiples ciclos de pruebas.
5) **Deployment:** Se realiza el despliegue y la entrega de la aplicación con la funcionalidad volátil.

La ejecución de estas etapas da como resultado la obtención de código fuente, documentación y demás modelos que describen la funcionalidad volátil introducida.

Lógicamente el desarrollo de estas etapas para una implementación Ad-hoc y una con VF Framework variará uno de otro y deberá tener diferentes consideraciones al llevarse a cabo.

Una aplicación en la que una funcionalidad volátil es implementada de manera tradicional o ad-hoc, integra el código de la misma a la aplicación original, modificando así todas sus capas. Estas modificaciones están visibles durante toda la ejecución de la aplicación.

Con VF Framework, en cambio, las funcionalidades volátiles son implementadas de manera desacoplada, por ende la aplicación original no es modificada. Además, la visibilidad de las funcionalidades volátiles puede ser manipulada en tiempo de ejecución mediante la programación de eventos. De esta manera, una aplicación web puede desplegarse con una funcionalidad volátil desactivada, y luego ser activada mediante un evento de activación configurado con antelación.

El hecho de poder efectuar esta tarea de manera programada resulta realmente muy útil. Gracias a ello, VF Framework no sólo mejora y optimiza los resultados para el tratamiento de funcionalidades volátiles, sino que también lo automatiza.

5.2.2 Remoción de funcionalidades volátiles

El hecho de que las funcionalidades incorporadas sean volátiles determina que su presencia en aplicaciones web sea sólo por un tiempo finito. Pasado ese tiempo resulta necesario eliminarlas. Esta remoción también forma parte del ciclo de vida de las funcionalidades volátiles y afecta la etapa de mantenimiento del ciclo de vida de un desarrollo de software, requiriendo un mantenimiento de perfeccionamiento.

Las etapas que conlleva esta remoción son las mismas que las mencionadas para la incorporación. Sin embargo, se diferencia de ella, dado que aquí las etapas de Análisis y Diseño se ven simplificadas gracias a la posibilidad de reutilizar la documentación y los modelos descriptivos obtenidos previamente durante la incorporación. Por esta razón, se hará foco en el resto de las etapas (Implementación, Testing y Deployment).

A continuación, se mostrarán las diferencias existentes entre una implementación Ad-hoc y una implementación VF Framework a la hora de remover una funcionalidad volátil.

Implementación Ad-hoc

El ciclo de vida de una funcionalidad volátil que es removida de una aplicación web de manera ad-hoc involucra las siguientes etapas:

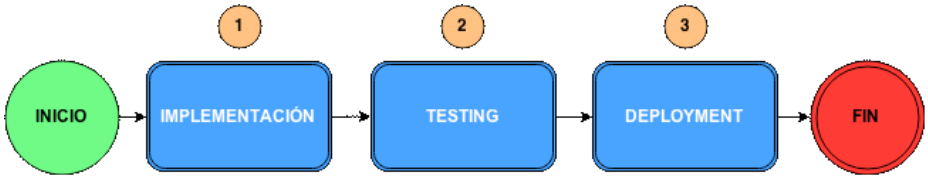


Ilustración 8: Etapas de remoción de FV en implementación Ad-hoc

1) **Implementación:** Se codifica y se prueban los cambios introducidos mediante tests de código y de unidad. Para eliminar la funcionalidad volátil se deberán remover manualmente todos los cambios que han sido introducidos previamente. Si desde la versión anterior de la aplicación a la versión actual de la misma solamente se ha agregado código para introducir la funcionalidad volátil, entonces su remoción será tan simple como volver a la versión anterior. Sin embargo, si posteriormente se han realizado otros cambios en el medio, la tarea será mucho más dificultosa. Esta tarea puede generar problemas como la disminución de calidad del código y la introducción de errores.
2) **Testing:** Se realizan principalmente pruebas de integración y regresión para asegurar el correcto funcionamiento de la aplicación. Esta etapa suele requerir la ejecución de múltiples ciclos de pruebas.
3) **Deployment:** Se realiza el despliegue y la entrega de la aplicación sin la funcionalidad volátil.

Implementación VF Framework

VF Framework brinda la posibilidad de efectuar la desactivación de funcionalidades volátiles en tiempo de ejecución. A diferencia de lo que ocurre con una implementación de tipo ad-hoc de una funcionalidad volátil, luego de ejecutarse el deployment de una aplicación no es necesario detenerla para hacer desaparecer la funcionalidad volátil. Mediante la configuración de un evento de desactivación, una funcionalidad volátil puede ser desactivada en tiempo de ejecución. La aplicación sólo requeriría ser detenida en caso de querer remover una funcionalidad volátil, no mediante su desactivación sino de manera definitiva.

El ciclo de vida de una funcionalidad volátil que es removida de manera definitiva de una aplicación web según VF Framework posee el siguiente diagrama de flujo:

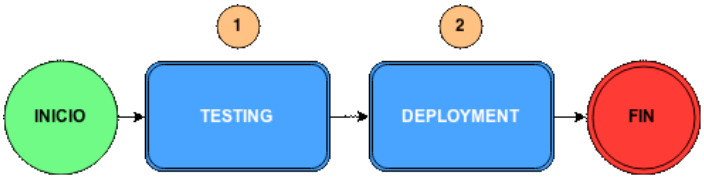


Ilustración 9: Etapas de remoción de FV en implementación VF Framework

1) **Testing:** Se realizan principalmente pruebas de integración y regresión para asegurar el correcto funcionamiento de la aplicación. Esta etapa suele requerir la ejecución de un único ciclo de pruebas. La simple prueba de la aplicación original (que no ha sufrido ningún cambio) debe ser suficiente para conseguir el objetivo deseado.
2) **Deployment:** Se realiza el despliegue y la entrega de la aplicación original sin la funcionalidad volátil.

Como puede observarse, la etapa "Implementación" aquí ha desaparecido. Esto se debe a que en caso de utilizar este marco de trabajo, efectivamente no es necesario realizar una implementación de ningún tipo para la remoción de funcionalidades volátiles. La aplicación original nunca es modificada, por lo cual su ejecución debería bastar para cumplir con el objetivo planteado. Además, de esta manera se asegura evitar la generación de problemas en la etapa de mantenimiento tales como la disminución de la calidad del código y la introducción de errores, sumados al aumento de esfuerzo de trabajo, de tiempos y de costos.

5.2.3 Reincorporación de funcionalidades volátiles

Así como las funcionalidades volátiles aparecen en un primer momento y luego son removidos, en el futuro, eventualmente, pueden ser precisadas nuevamente. Esta reincorporación es opcional, por lo cual no siempre aparece dentro del ciclo de vida de una funcionalidad volátil en una aplicación web. Por supuesto la reincorporación de una funcionalidad volátil requiere un esfuerzo mucho menor que la incorporación de una nueva funcionalidad volátil. Sin embargo, esta reincorporación también afecta la etapa de mantenimiento del ciclo de vida de un desarrollo de software, requiriendo un mantenimiento de perfeccionamiento.

Al igual que para la remoción, las etapas de Análisis y Diseño se ven simplificadas debido a la reutilización de documentación y modelos descriptivos

desarrollados durante la incorporación en un primer momento. Por esta razón, se hará foco en el resto de las etapas (Implementación, Testing y Deployment).

A continuación, se mostrarán las diferencias existentes entre una implementación Ad-hoc y una implementación VF Framework a la hora de reincorporar una funcionalidad volátil.

Implementación Ad-hoc

El ciclo de vida de una funcionalidad volátil que es reincorporada de una aplicación web de manera ad-hoc involucra las siguientes etapas:

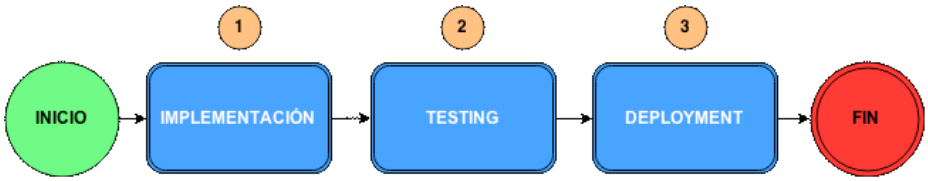


Ilustración 10: Etapas en reincorporación de FV en implementación Ad-hoc

1) **Implementación:** Se codifica y se prueban los cambios introducidos mediante tests de código y de unidad. Para reincorporar la funcionalidad volátil se deberán reincorporar manualmente todos los cambios que han sido eliminados previamente. Si desde la versión anterior de la aplicación a la versión actual de la misma solamente se ha eliminado código para remover la funcionalidad volátil, entonces su reincorporación será tan simple como volver a la versión anterior. Sin embargo, si posteriormente se han realizado otros cambios en el medio, la tarea será mucho más dificultosa. Esta tarea puede generar problemas como la disminución de calidad del código y la introducción de errores.
2) **Testing:** Se realizan principalmente pruebas de integración y regresión para asegurar el correcto funcionamiento de la aplicación. Esta etapa suele requerir la ejecución de múltiples ciclos de pruebas.
3) **Deployment:** Se realiza el despliegue y la entrega de la aplicación con la funcionalidad volátil visible durante toda su ejecución.

Implementación VF Framework

VF Framework también brinda la posibilidad de efectuar la activación de funcionalidades volátiles en tiempo de ejecución. Si la funcionalidad volátil ha sido incorporada previamente pero se encuentra desactivada, entonces no es necesario detener la aplicación para hacer reaparecer la funcionalidad volátil. Mediante la configuración de un evento de activación, una funcionalidad volátil puede ser activada en tiempo de ejecución. En cambio, si la funcionalidad volátil ha sido removida de manera definitiva, entonces sí resulta necesario detener la aplicación.

El ciclo de vida de una funcionalidad volátil que es reincorporada (luego de ser removida de manera definitiva) en una aplicación web según VF Framework posee el siguiente diagrama de flujo:

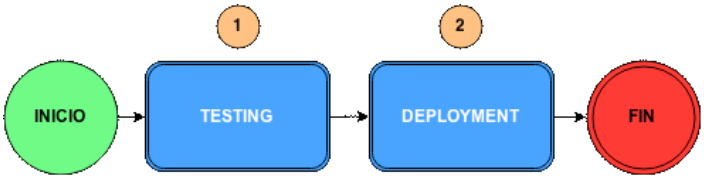


Ilustración 11: Etapas en reincorporación de FV en implementación VF Framework

1) **Testing:** Se realizan principalmente pruebas de integración y regresión para asegurar el correcto funcionamiento de la aplicación. Esta etapa suele requerir la ejecución de un único ciclo de pruebas. La simple prueba de la aplicación original decorada con la funcionalidad volátil deseada (implementada previamente) debe ser suficiente para conseguir el objetivo deseado.
2) **Deployment:** Se realiza el despliegue y la entrega de la aplicación original con la funcionalidad volátil.

Como puede observarse, la etapa "Implementación" aquí ha desaparecido. Esto se debe a que, al igual que para la remoción, en caso de utilizar este marco de trabajo, efectivamente no es necesario realizar una implementación de ningún tipo para la reincorporación de funcionalidades volátiles. La ejecución de la aplicación original decorada por la funcionalidad volátil deseada (implementada previamente) debería bastar para cumplir con el objetivo planteado. Además, de esta manera se asegura evitar la generación de problemas en la etapa de mantenimiento tales como la disminución de la calidad del código y la introducción de errores, sumados al aumento de esfuerzo de trabajo, de tiempos y de costos.

5.3 Ejemplificación

A lo largo del capítulo tomaremos el siguiente caso como ejemplo para poder brindar explicaciones claras y concretas acerca de los temas subyacentes al VF Framework. Con este propósito, dicho caso podrá ser modificado o se podrán considerar nuevas restricciones en las siguientes secciones.

Una empresa cuenta con un sitio web dedicado a la comercialización online de productos. Esta empresa ha decidido tomar ciertas medidas con el objetivo de impulsar sus ventas:

- **Lanzamiento de descuentos (DISCOUNT VF):** La empresa ha lanzado de manera temporal una promoción que consta de importantes descuentos para algunos de sus productos. Téngase en cuenta que los descuentos agregados no son para todos los productos, sino sólo para un subconjunto de ellos. Además, el monto a descontar para los productos en promoción es totalmente independiente y puede variar de un producto a otro. El registro de las ventas realizadas deberá contar con el descuento realizado para cada uno de los productos involucrados en las ventas.
- **Regalos especiales (GIFT VF):** La empresa ha decidido premiar a aquellos compradores que realicen compras con montos superiores a los $1000. Por cada compra que supere dicho monto, se hará entrega de un regalo especial. Deberá guardarse el registro acerca de las ventas que han incluido regalos. Esta promoción se mantendrá hasta agotar el stock de los regalos.

A continuación, en la Ilustración 12, se mostrará el diagrama UML con las clases principales de la aplicación web.

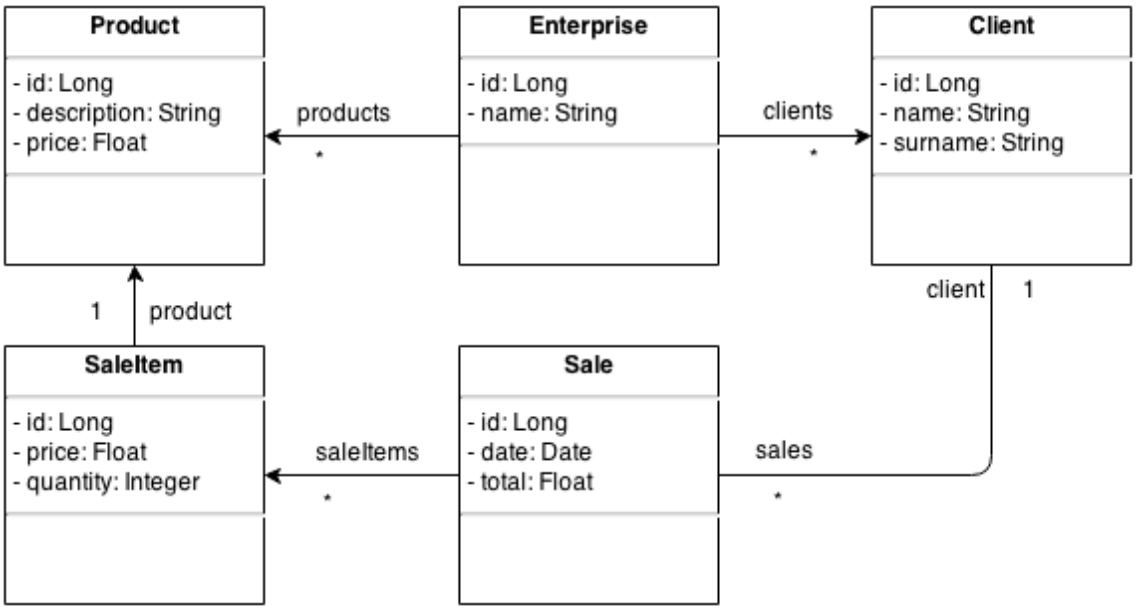


Ilustración 12: Diagrama UML de la aplicación de ejemplo sin FVs

Por otro lado, en la Ilustración 13, se mostrará el diagrama UML de la aplicación considerando los cambios introducidos por las funcionalidades volátiles "DISCOUNT VF" y "GIFT VF".

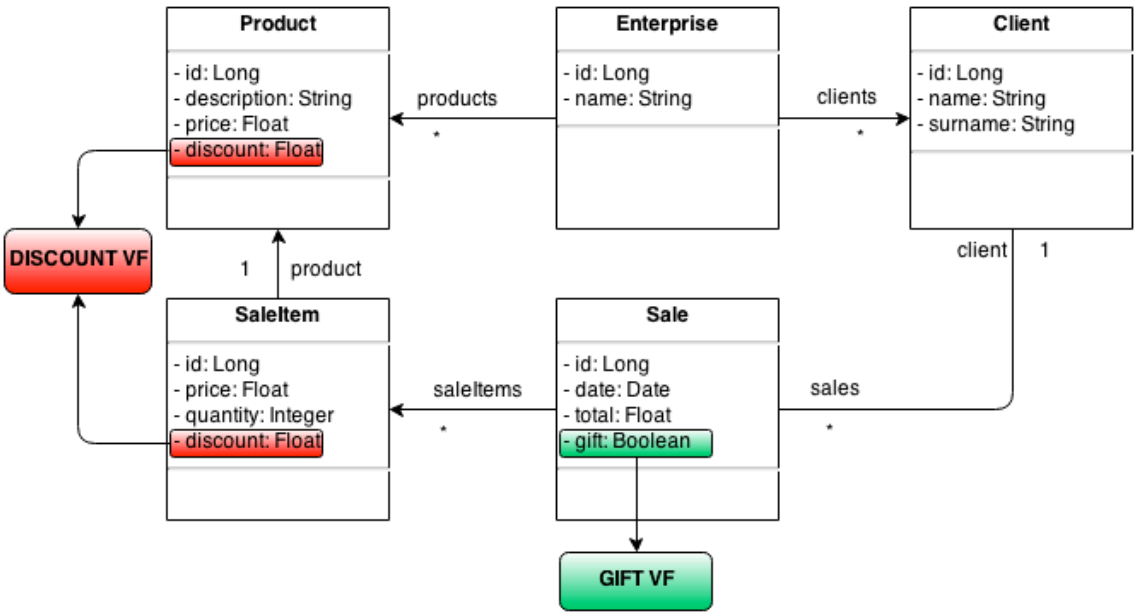


Ilustración 13: Diagrama UML de la aplicación de ejemplo con FVs

Justificación de decisión de diseño: Teniendo en cuenta que para un producto puede modificarse el valor de su precio (y de su descuento si la funcionalidad volátil está activada), se ha decidido modelar la clase SaleItem y que la misma también almacene esta información (precio y descuento). De esta forma, aunque los datos de un

producto sean modificados, los datos de los registros de ventas efectuadas no se verán afectados.

5.4 Requerimientos necesarios

En esta sección, se describirán los requerimientos fundamentales con los que debe contar un lenguaje de programación y que deben ser tenidos en cuenta para poder añadir funcionalidades volátiles haciendo uso de este Framework. Si el lenguaje de programación escogido es capaz de soportarlos, entonces será capaz de implementar una aplicación según los conceptos del VF Framework. Este marco de trabajo fue desarrollado para lenguajes de programación que siguen el paradigma de orientación a objetos.

Tal como se discute en [10], los puntos más importantes que el Framework propuesto debe cubrir pueden ser agrupados en:

- Arquitectura desacoplada
- Modificación de las capas de la aplicación
- Gestión de las modificaciones

Se propondrán posibles implementaciones para llevar a cabo cada uno de estos puntos. La elección final dependerá por un lado de la decisión del desarrollador, y por otro lado de las características, posibilidades y limitaciones propias del lenguaje de programación seleccionado.

5.4.1 Arquitectura desacoplada

La arquitectura desacoplada es uno de los puntos fundamentales para el marco de trabajo presentado. Con una arquitectura desacoplada se hace referencia a la separación de la implementación de las funcionalidades volátiles de la implementación de la aplicación original. Debe poder mantenerse por un lado el código que forma parte de la aplicación original, y por otro todo el código correspondiente a las funcionalidades volátiles y a su gestión. Como ya ha sido explicado, esta separación proporciona ventajas tales como la preservación de la integridad de la aplicación original, la mejora del mantenimiento y la mayor modularización y reutilización de código.

5.4.2 Modificación de las capas de la aplicación

La introducción de funcionalidades volátiles en una aplicación web involucra la modificación de cada una de sus capas. Cuando una funcionalidad volátil es activada o desactivada, el cambio debe verse reflejado en todas las capas de la aplicación.

Consecuentemente, una funcionalidad volátil está compuesta por:

- Modificación de la capa de negocio
- Modificación de la capa de navegación
- Modificación de la capa de presentación

5.4.2.1 Modificación de la capa de negocio

La modificación de la capa de negocio (también llamada capa de modelo) incluye cuestiones tales como:

- Inyección de variables y relaciones

- Mapeo de variables y relaciones
- Inyección de métodos
- Modificación del flujo original de métodos existentes

Inyección de variables y relaciones: La inyección de variables y relaciones implica cambios en las clases pertenecientes al modelo de la aplicación web. Este punto resulta indispensable para poder extender la estructura que podría requerir una funcionalidad volátil. Las relaciones podrían involucrar tanto clases preexistentes en el modelo de datos, como también clases nuevas añadidas por la funcionalidad volátil en cuestión.

Como puede observarse en la Ilustración 14, en el ejemplo, al activarse la funcionalidad volátil "DISCOUNT VF", se le incorpora la variable de instancia "discount" a la clase "Product".

Ilustración 14: Inyección de variable

Mapeo de variables y relaciones: La inyección de variables y relaciones sirve para modificar la estructura de una clase; no obstante, esto puede resultar insuficiente. Dichas modificaciones introducidas posiblemente involucren la aparición de nuevos datos relevantes que deben ser almacenados en la base de datos utilizada. Para ello, es necesario definir y ejecutar los cambios correspondientes en el mapeo de las entidades involucradas. De esta manera, el ORM utilizado puede efectuar el mapeo de los atributos y relaciones de los objetos de la clase modificada a la base de datos relacional.

Como puede observarse en la Ilustración 15, en el ejemplo, al activarse la funcionalidad volátil "DISCOUNT VF", se le incorpora la columna "discount" a la tabla "Product".

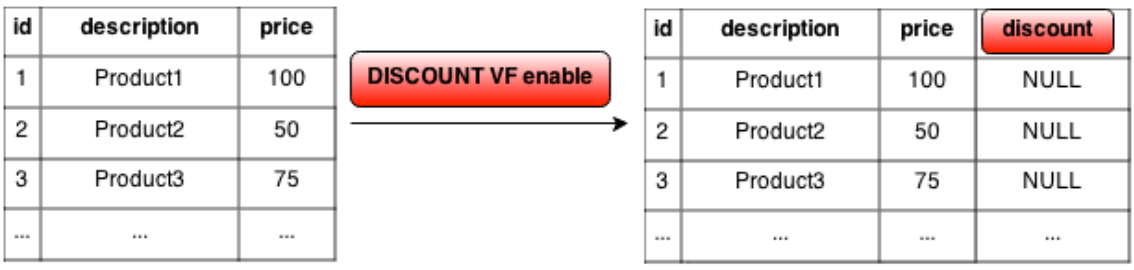


Ilustración 15: Mapeo de variable

Inyección de métodos: Adición de nuevos métodos a clases ya existentes, extendiendo el comportamiento de las mismas. Puede utilizarse tanto para la definición de métodos getters y setters para las nuevas variables introducidas como para la definición de cualquier tipo de método nuevo.

Como puede observarse en la Ilustración 16, en el ejemplo, al activarse la funcionalidad volátil "DISCOUNT VF", se incorporan métodos de instancia getter y setter para la variable de instancia "discount". Además, se incorpora el método de instancia #getFinalPrice(), mediante el cual puede obtenerse el precio final de un producto teniendo en cuenta su correspondiente descuento.

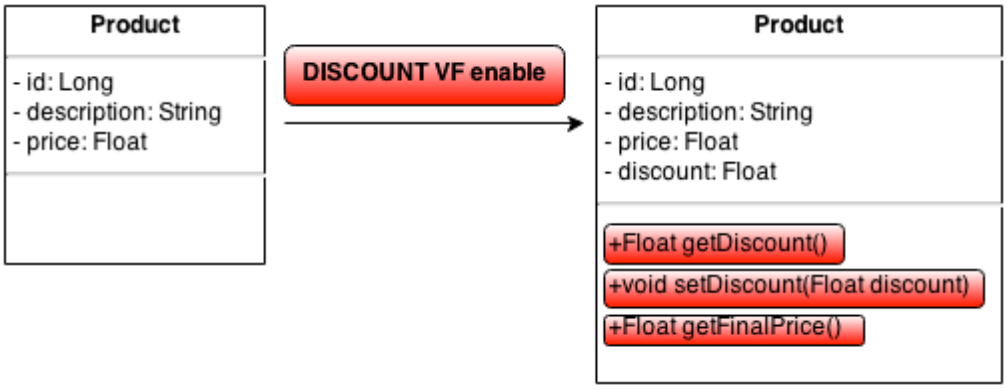


Ilustración 16: Inyección de métodos

Modificación del flujo original de métodos existentes: La introducción de funcionalidades volátiles en una aplicación web puede requerir la modificación del flujo original de determinados métodos existentes para su correcto acoplamiento y funcionamiento. Por esta razón, el Framework precisa brindar la posibilidad de alterar el comportamiento preexistente de clases. Esto puede lograrse a partir de dos mecanismos:

- **Intercepción de métodos:** La intercepción de métodos consta de la definición de dos cuestiones. Primero, deben definirse los puntos específicos de la aplicación en los cuales se desea modificar el flujo original de la misma. Esto se realiza indicando el o los métodos a interceptar. Luego, se deben definir las acciones a efectuar ante la invocación de dichos puntos indicados previamente.
- **Modificación de métodos:** La modificación de métodos hace referencia a la modificación del código fuente original de los mismos. Esta alternativa es una posibilidad si y sólo si se cuenta con un lenguaje de programación dinámicamente tipado capaz de generar código dinámicamente. Además, debe contar con algún mecanismo automatizado que permita que el código fuente original de los métodos a modificar pueda ser almacenado, para luego ser restaurado cuando corresponda. De no ser así, se estaría violando uno de los puntos más importantes del framework, que es la implementación de una arquitectura desacoplada.

En el ejemplo, al activarse la funcionalidad volátil "DISCOUNT VF", deberían interceptarse los puntos en los cuales se obtienen los precios de los productos seleccionados con el propósito de sumarlos para calcular el valor total de una venta. El total de la misma debería ser calculado a partir de los precios finales, y no de los precios

originales. Por ese motivo, deberían efectuarse las intercepciones y/o modificaciones de los métodos pertinentes.

Programación orientada a aspectos (AOP)

La programación orientada a aspectos resulta fundamental para cubrir los puntos que acaban de ser mencionados para las modificaciones de la capa de negocio. Sirve para inyectar variables y métodos y para modificar el accionar de métodos mediante la intercepción de los mismos.

Deberá dársele especial énfasis al tipado que posea el lenguaje de programación elegido, dado que de él dependerá la estrategia para tratar las modificaciones de la capa de negocio. A continuación se presenta dicha clasificación:

- **AOP para lenguajes de programación estáticamente tipados:** Incluye la incorporación de variables, relaciones y métodos nuevos, e intercepción de métodos mediante la definición de conjuntos de pointcuts y advices asociados. Al iniciarse la aplicación, se incorporan todos los componentes mencionados. La modificación del flujo de métodos existentes sólo puede realizarse a partir de la intercepción de métodos; y dado que esta intercepción es definida estáticamente en tiempo de compilación, la misma siempre estará presente en el código fuente compilado. Por esta razón, la ejecución de las intercepciones de métodos debe ser controlada mediante el uso de sentencias condicionales basadas en el estado de determinada funcionalidad volátil. Las intercepciones deberían ser efectuadas si y sólo si el estado de la funcionalidad volátil de la que dependen es activado. De esta manera, se logra que la clase en cuestión pueda accionar con o sin el cambio de comportamiento incorporado según corresponda. Los nuevos métodos inyectados no requieren la consulta del estado de una funcionalidad volátil, dado que cuando ésta se encuentra desactivada ni siquiera deberían ser invocados.
- **AOP para lenguajes de programación dinámicamente tipados:** Incluye la incorporación y remoción dinámica de métodos, variables e intercepciones de métodos según corresponda a partir de la activación y desactivación de funcionalidades volátiles. Mientras que para lenguajes estáticamente tipados sólo existe la opción de resolverlo durante tiempo de compilación, en los lenguajes dinámicamente tipados las posibilidades son más amplias. En ellos, los aspectos se diferencian en que también pueden ser instalados y desinstalados en tiempo de ejecución. La instalación de un aspecto implica la incorporación de todas sus modificaciones definidas. Por otro lado, la desinstalación de un aspecto implica la remoción de todas sus modificaciones definidas. Por esta razón, los métodos interceptados no requieren la consulta del estado de determinada funcionalidad volátil para decidir si deben ser ejecutados o no. Por el contrario, los cambios establecidos por el aspecto sólo estarán presentes en caso de estar activada la funcionalidad volátil correspondiente.

Modificaciones Regex

La intercepción de métodos mediante aspectos es un factor muy importante a la hora de modificar el comportamiento de métodos. Sin embargo, cuando deben coexistir

múltiples funcionalidades volátiles que involucran modificaciones de secciones de código coincidentes, la intercepción puede no ser suficiente. Los advices pueden ser definidos para ejecutarse antes, después o en el momento de acceder a determinado pointcut. La correcta ejecución de un método interceptado por múltiples funcionalidades volátiles está limitada por el grado de modularización que posean los métodos de las clases de la aplicación.

Los lenguajes estáticamente tipados podrían requerir elevar el grado de modularización de métodos si varias funcionalidades volátiles involucran intercepciones en métodos convergentes.

Los lenguajes dinámicamente tipados, en cambio, pueden tratar este inconveniente de una mejor manera. Los aspectos pueden ser combinados con la modificación de métodos a través de expresiones regulares.

Mediante la utilización de expresiones regulares pueden identificarse con una eficaz precisión sectores del código fuente de un método. De esta manera, luego de identificarlos, se los puede modificar en tiempo de ejecución. Múltiples funcionalidades pueden activarse modificando al mismo método sin limitación alguna. No obstante, hay que prestar especial atención cuando se desactiva una funcionalidad volátil que ha modificado un método, que a su vez también ha sido modificado por otras funcionalidades volátiles. En ese caso, dicho método debería volver a su estado original y volver a aplicar las modificaciones correspondientes a las funcionalidades volátiles que aún se encuentran activadas.

Las modificaciones regex pueden ser utilizadas sólo para lenguajes dinámicamente tipados dado que se requiere la generación dinámica de código.

5.4.2.2 Modificación de la capa de navegación

La modificación de la capa de navegación involucra la alteración tanto de nodos como de enlaces navegacionales, con el fin de lograr una correcta adaptación a los contextos navegacionales deseados. Las modificaciones de la capa de navegación involucran todos aquellos cambios relacionados con la capa Controller de una aplicación MVC. Es decir, principalmente cambios que afecten a controladores y a servicios y demás clases que sirvan como apoyo para esos controladores.

Entre ellos se incluyen:

- **Inyección de dependencias:** La inyección de dependencias implica cambios similares a la inyección de atributos y relaciones en la capa de modelo, sólo que aquí lo que se inyectan son otros componentes como servicios y clases helpers. Este punto resulta importante ya que puede ser necesario el consumo de nuevas dependencias con el fin de modificar el comportamiento de determinada clase perteneciente a la capa de navegación. Además, las dependencias inyectadas podrían involucrar tanto clases preexistentes como también clases nuevas añadidas por la funcionalidad volátil en cuestión.

  En el ejemplo, consideremos la existencia de una clase llamada “SaleController” para la manipulación del módulo de ventas. Al activarse la funcionalidad volátil “GIFT VF”, la clase “SaleController” podría incorporar una referencia a un helper “GiftHelper”. En el helper “GiftHelper” se podría implementar la lógica de validación para definir si una compra debe incluir un regalo.
- **Inyección de métodos:** La inyección de métodos también implica cambios similares a la inyección de métodos perteneciente a la capa de

modelo. Contempla la adición de nuevos métodos a clases ya existentes, extendiendo el comportamiento de las mismas. Puede utilizarse tanto para la definición de métodos getters y setters para las dependencias inyectadas como para la definición de cualquier tipo de método nuevo, como firmas nuevas en interfaces de servicios, o métodos que definan cierta lógica nueva de validación necesaria.
En el ejemplo, al activarse la funcionalidad volátil "GIFT VF", la clase "SaleController" requeriría la inyección de métodos getter y setter para el helper "GiftHelper".

- **Modificación del flujo original de métodos existentes:** También al igual que en la modificación de la capa de negocio, la modificación del flujo original de métodos existentes puede darse por intercepción y por modificación de métodos. Este punto es esencial para lograr la modificación de la lógica de ejecución de métodos existentes.
  En el ejemplo, al activarse la funcionalidad volátil "GIFT VF", la clase "SaleController" requeriría modificar su lógica de ejecución al generar nuevas ventas. Este proceso debería incluir la consulta del helper "GiftHelper" para definir el valor que debe ser seteado en la variable "gift".

5.4.2.3 Modificación de la capa de presentación

La modificación de la capa de presentación implica la alteración de las vistas que comprenden la interfaz de la aplicación web. Una vez que las modificaciones del modelo han sido consumadas, resulta fundamental que estas se vean reflejadas en la interfaz que consumirá el usuario final.

Esta tarea consiste en efectuar las transformaciones correspondientes a cada vista y su visualización según el estado (activado o desactivado) en que se encuentren las funcionalidades volátiles incluidas en la aplicación. La aplicación resultante debe ser capaz de mostrar u ocultar los cambios introducidos en tiempo de ejecución.

Se han encontrado dos enfoques para resolver este tipo de modificaciones:

- Visualización condicional de modificaciones
- Modificaciones en tiempo de ejecución

Visualización condicional de modificaciones: La visualización condicional de modificaciones implica la transformación de archivos asociados a las vistas de la aplicación mediante la introducción de sentencias condicionales. Estas sentencias condicionales (presentes durante toda la ejecución de la aplicación) son las responsables de visualizar o no el contenido de cada una de las modificaciones pertenecientes a las funcionalidades volátiles. Lógicamente, para que esto sea posible es necesario que los archivos de las vistas tengan la capacidad de procesar información del lado del servidor y puedan tomar decisiones en base a ella. En ese caso, se evalúa el estado en que se encuentra cada funcionalidad volátil en el momento en que una vista modificada es invocada. Si la funcionalidad volátil está activada, todos los cambios introducidos en la vista podrán ser observados. De lo contrario, se mostrará la vista original. Mediante esta solución las transformaciones deberían ejecutarse al momento de incorporar las funcionalidades volátiles. Luego de ejecutarse la transformación se podrá evaluar si visualizar los cambios o no.

En la Ilustración 17 se muestra el código y el resultado de una posible implementación de visualización condicional de modificaciones para el ejemplo dado.

Como puede observarse, el banner que informa la promoción con el texto “Aproveche nuestros importantes descuentos!!!” se visualiza sólo cuando la funcionalidad volátil “DISCOUNT VF” está activada. La visibilidad de este banner se resuelve del lado del servidor y depende del valor contenido en el flag “enabled” correspondiente a dicha funcionalidad volátil. Cabe destacar que esta sentencia condicional que lo define siempre estará presente y será evaluada del lado del servidor.

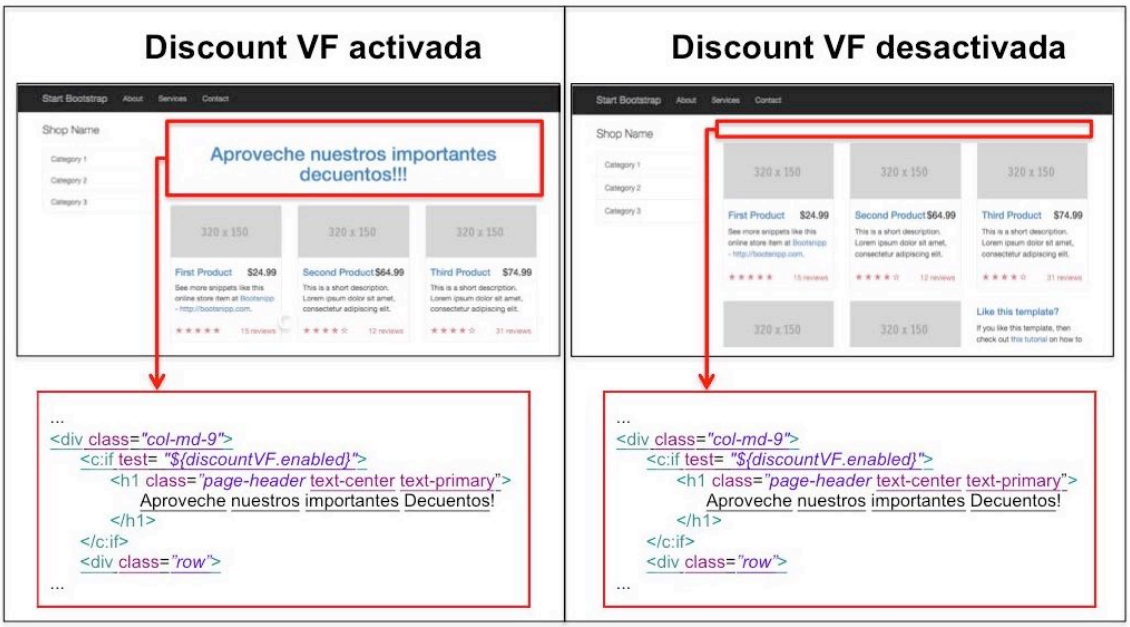


Ilustración 17: Visualización condicional de modificaciones

Modificaciones en tiempo de ejecución: Las modificaciones en tiempo de ejecución representan una solución más dinámica. Si se escogiera esta opción, las vistas no contarían con sentencias de condición encargadas de determinar la visualización o no de las modificaciones de la capa de presentación. Tampoco sería necesaria la ejecución de las transformaciones al momento de incorporar las funcionalidades volátiles a la aplicación. Por el contrario, cada vez que una funcionalidad volátil fuera activada o desactivada, se deberían modificar las vistas comprometidas en tiempo de ejecución. Esta solución requiere la preservación de una copia de la vista original, disponible para ser modificada cada vez que fuera necesaria. Se debe considerar también que el hecho de ejecutar las transformaciones en tiempo de ejecución puede resultar lento e ineficiente si no se cuenta con una arquitectura con la velocidad de procesamiento adecuada.

En la Ilustración 18 se muestra el código y el resultado de una posible implementación de modificaciones en tiempo de ejecución para el ejemplo dado. Al igual que en la Ilustración anterior, el banner que informa la promoción es visualizado sólo cuando la funcionalidad volátil “DISCOUNT VF” se encuentra activada. Sin embargo, se diferencian en que aquí no existen sentencias condicionales, sino que simplemente se visualiza el código que está presente en la vista. Esto es así dado que cuando la funcionalidad volátil “DISCOUNT VF” es activada, el código correspondiente al banner es añadido. Y por el contrario, cuando la funcionalidad volátil “DISCOUNT VF” es desactivada, el código correspondiente al banner es removido.

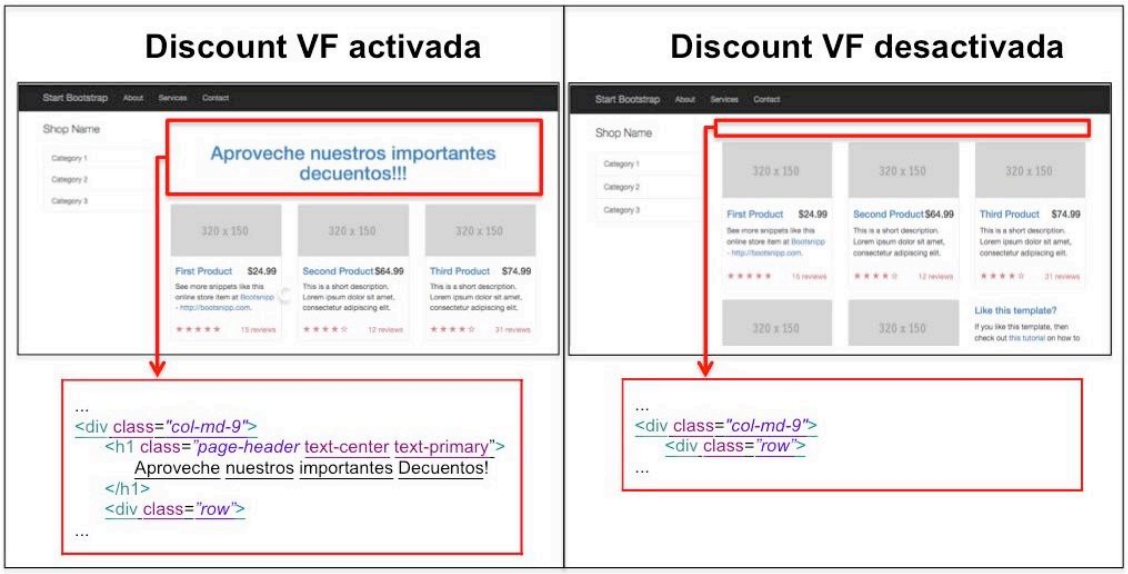


Ilustración 18: Modificaciones en tiempo de ejecución

5.4.3 Gestión de las modificaciones

Todas las modificaciones deben poder ser gestionadas en tiempo de ejecución. La gestión de estas modificaciones involucra:

- Administración centralizada de las funcionalidades volátiles
- Weaving de las funcionalidades volátiles
- Subetapas de las funcionalidades volátiles
- Commons para las funcionalidades volátiles
- Programación de eventos de activación y desactivación

5.4.3.1 Administración centralizada de las funcionalidades volátiles

La gestión de las funcionalidades volátiles definidas para una aplicación requiere la centralización de la administración de las mismas. De nada sirven las funcionalidades volátiles si no pueden ser manipuladas. Por esta razón, se ha definido una entidad encargada de manipularlas. Debe ser el único punto de entrada mediante el cual esto sea posible. Para ello, resulta adecuada la utilización del patrón de diseño Singleton descripto en [20]. Llamaremos a esta entidad que representa el núcleo del marco de trabajo “VFFramework”. La entidad VFFramework debe tener conocimiento de todas las funcionalidades volátiles incorporadas en la aplicación. Debe contar con operaciones para la activación, la desactivación y la consulta de estado de funcionalidades volátiles. Los lenguajes dinámicamente tipados contarán también con operaciones extra para el agregado y la remoción de funcionalidades volátiles en tiempo de ejecución.

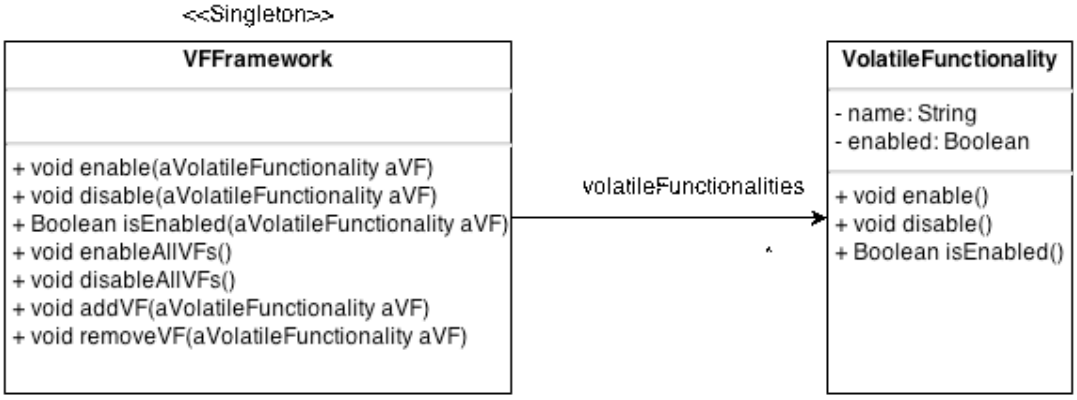


Ilustración 19: Administración centralizada de las FVs

5.4.3.2 Weaving de funcionalidades volátiles

El weaving de las funcionalidades volátiles se refiere al acoplamiento de las mismas con la aplicación original. Una vez que se han definido las funcionalidades volátiles en un nuevo proyecto, debe utilizarse algún mecanismo que permita lograr su integración con la aplicación original.

Estos mecanismos están directamente relacionados con el tipado del lenguaje de programación y pueden implicar dos tipos de weavings:

- **Weaving estático:** el weaving estático implica que la integración se da en tiempo de compilación. Este tipo de weaving es la única opción para lenguajes de programación estáticamente tipados. Generalmente involucra la definición de archivos de configuración. Si se utiliza weaving estático, una funcionalidad volátil puede ser desactivada en tiempo de ejecución, pero para removerla de manera definitiva es necesaria la recompilación de la aplicación.
- **Weaving dinámico:** el weaving dinámico implica que la integración se da en tiempo de ejecución. Este tipo de weaving solamente puede ser utilizado para lenguajes de programación dinámicamente tipados. El weaving dinámico permite tanto la desactivación como la remoción definitva de una funcionalidad volátil en tiempo de ejecución.

5.4.3.3 Subetapas de las funcionalidades volátiles

Al utilizar el VF Framework se obtiene una automatización de los procesos de activación y desactivación de las funcionalidades volátiles en una aplicación web. Sin embargo, para que dicho objetivo sea alcanzado, en ocasiones ambas acciones pueden requerir la definición de determinadas acciones complementarias. La actualización de datos de las base de datos es un claro y frecuente ejemplo de ello.

Cuando se procede a activar o desactivar una funcionalidad volátil que involucra la adición de un atributo a una clase, debería tratarse asimismo la correspondiente columna de la tabla de la base de datos. Para ello, debería escogerse una estrategia adecuada para la manipulación de los datos, teniendo en cuenta las consecuencias que trae aparejadas.

La elección de la correcta estrategia para el tratamiento de la base de datos estará condicionada principalmente por dos factores:

- **La relevancia del valor de los datos añadidos por el nuevo atributo:** Esta cuestión dependerá del caso particular y deberá ser analizada para poder tomarse una decisión respecto a la conservación y la integridad de los datos. Deberá definirse si los datos en cuestión son de interés o no para el propietario de los mismos.
- **El tipo de entidad a tratar:** El tipo de entidad a tratar puede ser clasificado en los siguientes dos tipos:
    - **Entidad con datos estáticos:** Este tipo de entidad incluye a aquellas entidades en las cuales una vez que una de sus instancias ha sido persistida, la misma no es modificada en el futuro. Este tipo de datos son almacenados generalmente como registros de alguna transacción y no suelen ser tomados como referencia por otra entidad. Para este tipo de entidades, el valor de los datos añadidos por un nuevo atributo suele tener gran relevancia.
      En el ejemplo, los items correspondientes a las ventas realizadas representan un claro caso de una entidad con datos estáticos. En ellos sería lógico que interese registrar tanto el precio, como el descuento del producto aplicados al momento de la venta. Por esta razón, todos los datos deben ser resguardados. Tal como se muestra en la Ilustración 20, dichos datos son almacenados en la tabla "SaleItem". Esta tabla está formada por las columnas: "id" (identificador del item), "sale_id" (identificador de la venta), "product_id" (identificador del producto), "price" (precio del producto), "quantity" (cantidad de productos) y "discount" (descuento del producto).

id	sale_id	product_id	price	quantity	discount
1	1	1	100	5	0
2	1	3	75	2	0
3	2	2	50	8	15
...	...	...	...	...	...

Ilustración 20: Tabla SaleItem

    - **Entidad con datos dinámicos:** Este tipo de entidad se refiere a entidades en las cuales una vez que una de sus instancias ha sido persistida, la misma sigue sufriendo modificaciones a lo largo del tiempo. Normalmente son utilizadas como referencia por otras entidades y su valor actual tiene una relevancia fundamental en el desarrollo del negocio.
      En el ejemplo, el listado de productos representa un claro caso de una entidad con datos dinámicos. En él interesa principalmente la información actual que posean las instancias de la entidad definida para los productos. Ésto es así dado que esta información será determinante para la obtención del precio final de cada uno de los productos involucrados en las ventas. Tal como se muestra

en la Ilustración 21, dichos datos son almacenados en la tabla "Product". Esta tabla está formada por las columnas: "id" (identificador del producto), "description" (descripción del producto), "price" (precio del producto) y "discount" (descuento del producto).

id	description	price	discount
1	Product1	100	0
2	Product2	50	15
3	Product3	75	0
...	...	...	...

Ilustración 21: Tabla Product

5.4.3.3.1 Inicialización de activación

La "Inicialización de Activación" (o "EnableSetUp" en inglés) es una etapa que ha sido definida para ejecutarse posteriormente a la activación de una funcionalidad volátil. Esto se ve reflejado de manera simple en la Ilustración 22. Se ha descubierto que luego de incorporar una funcionalidad volátil, es muy recurrente la necesidad de efectuar cierta inicialización o definición de valores para los nuevos atributos o relaciones añadidas a los datos ya existentes.

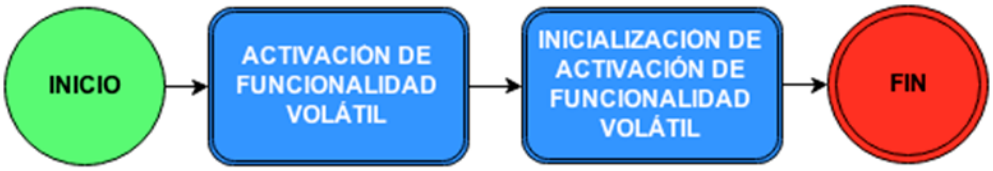


Ilustración 22: Inicialización de activación

Cuando se activa por primera vez una funcionalidad volátil, que incluye entre sus cambios la incorporación de un nuevo atributo en una entidad, esto tiene repercusión en la base de datos. La situación resultante es la siguiente:

- Una nueva columna es añadida a la tabla de la entidad.
- Todas las filas presentes en la tabla de la entidad poseen el valor correspondiente a dicha nueva columna en NULL.

Una columna agregada a una tabla preexistente con datos previamente cargados debe ser NULLABLE (es decir aceptar valores NULL). Esto es indispensable para que los datos presentes en la tabla queden con dicho atributo como NULL. De lo contrario, no sería posible mapear la columna sin antes desechar todos los datos contenidos en la tabla.

Por supuesto, el hecho de que estos valores queden en NULL puede afectar en ciertos casos al correcto funcionamiento de la aplicación web.

Si volvemos al ejemplo, una vez activada la funcionalidad de descuentos, el precio de un producto pasa a obtenerse a partir del cálculo resultante entre el precio original del producto y el descuento del mismo. Si se intentara calcular el precio total de un producto que tiene NULL como valor de descuento, lógicamente se produciría un error. Dicha situación se muestra en la Ilustración 23, en donde la tabla de la clase

“Product” incorpora la columna “discount” al activarse la funcionalidad volátil “DISCOUNT VF”.

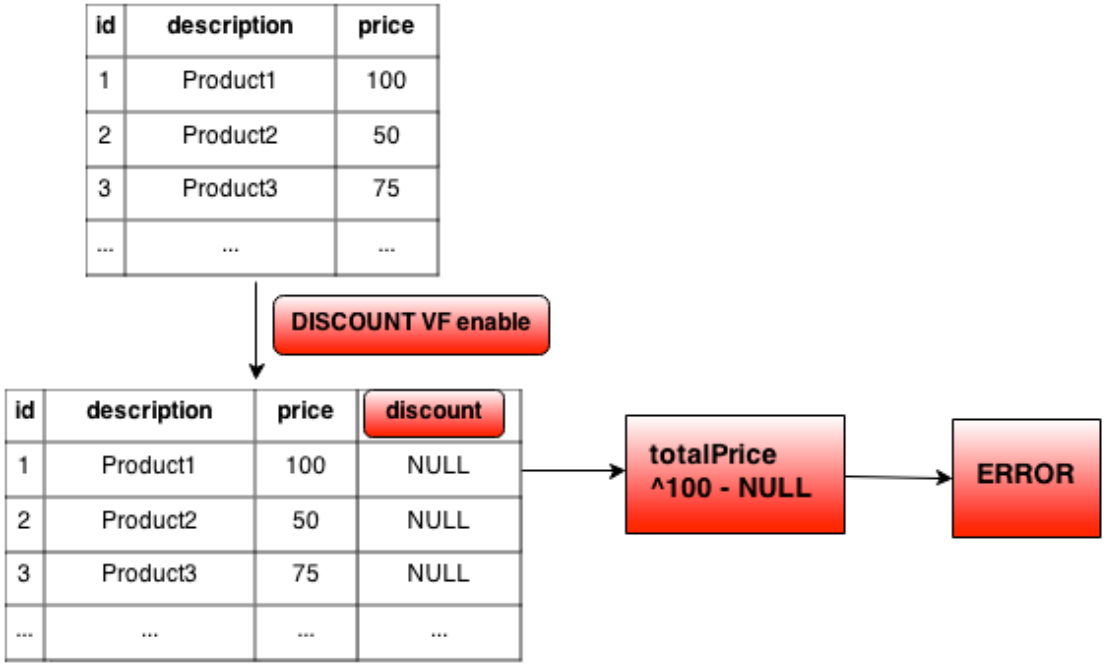


Ilustración 23: Problema en activación de FV

Este problema pudo ser resuelto mediante la definición de una nueva etapa que ha sido llamada “Inicialización de activación”.

En esta nueva etapa, deberían definirse todos los nuevos valores a asignar al atributo en cuestión. Tanto para los datos que han sido directamente afectados por la introducción de la nueva funcionalidad volátil, como para los que no. Los datos para los que no se han definido valores explícitamente deberían igualmente inicializar sus valores con algún valor por defecto. Además, este mismo valor también debería ser definido como valor por defecto para los nuevos productos que fueran incorporados al listado de productos de la empresa.

Esto ha sido aplicado y puede observarse en la Ilustración 24 correspondiente al ya mencionado ejemplo. En ella, el producto “Product2” se encuentra en promoción, habiendo sido inicializado con un descuento con el valor 15. No obstante, los productos “Product1” y “Product3” que no están en promoción, ya no poseen valor NULL sino que, como corresponde, han sido actualizados a 0.

Nota: En caso de ser necesario es posible definir un atributo como NOT-NULL (es decir denegar la inserción de valores nulos). Primero debería definirse el atributo como NULLABLE al incorporar la columna tal cual fue descripto anteriormente. Luego, en la etapa de “Inicialización de activación”, también deberían modificarse todos los valores de los datos preexistentes. Y finalmente, se debería modificar la columna para que sólo acepte valores NOT-NULL. Téngase en cuenta que si opta por utilizar esta opción, también requerirá su atención en la etapa de “PreDesactivación”, en donde se debería volver a definir el atributo como NULLABLE antes de desactivar la funcionalidad.

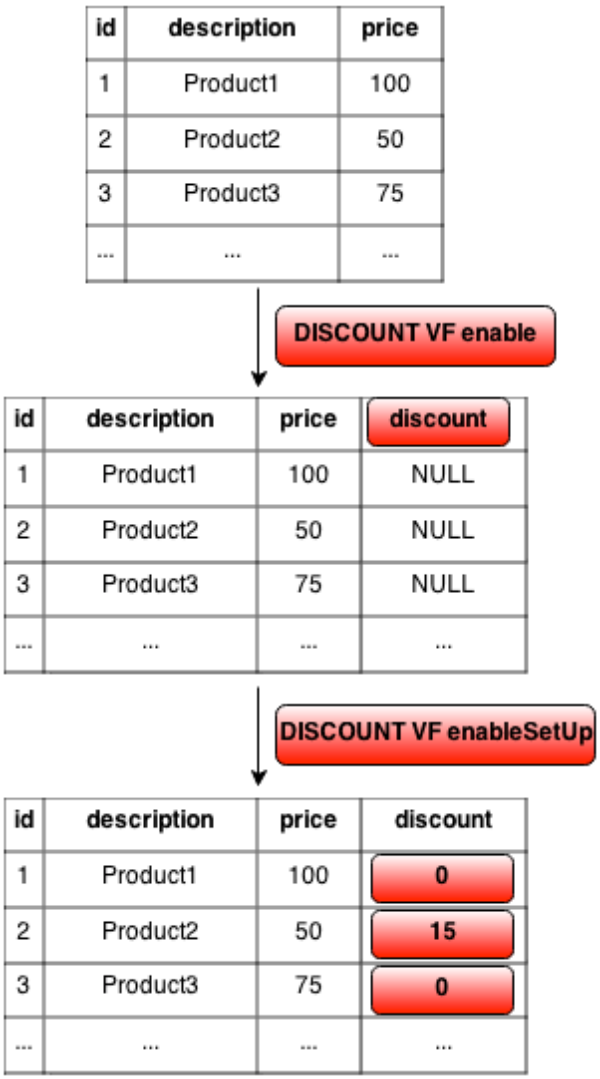


Ilustración 24: Solución en activación de FV con inicialización de activación

5.4.3.3.2 PreDesactivación

La **"**PreDesactivación" (o "PreDisable" en inglés) es una etapa que ha sido definida para ejecutarse previamente a la desactivación de una funcionalidad volátil. Esto se ve reflejado de manera simple en la Ilustración 25. A diferencia de lo que ocurre en la activación, aquí en ciertos casos se requiere la ejecución de acciones previas a la desactivación de una funcionalidad volátil. Esto podría involucrar cuestiones tales como la modificación, el tratamiento y el resguardo de datos de la base de datos.

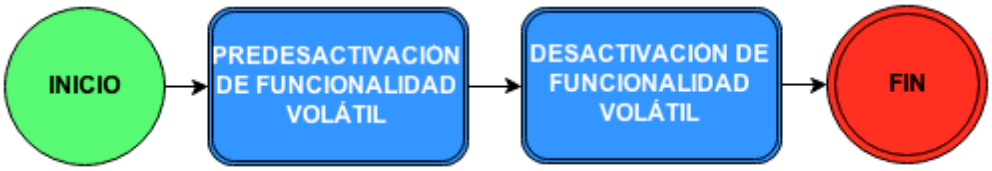


Ilustración 25: PreDesactivación

A continuación, se procederá a presentar algunas posibles estrategias a utilizar ante diferentes situaciones para llevar a cabo el tratamiento de datos luego de la desactivación de una funcionalidad volátil:

- **Eliminación de columna:** Esta estrategia consta simplemente de eliminar la columna que ha sido agregada previamente en la etapa de activación de una funcionalidad volátil. No obstante, tiene como consecuencia la pérdida de información que podría resultar valiosa en el futuro. Esta opción sólo debería ser tenida en cuenta si se está totalmente seguro de que la información representada por la columna en cuestión ya no tiene ninguna relevancia, ni la tendrá.
  Esta estrategia ha sido aplicada y puede observarse en la Ilustración 26 correspondiente al dominio del ejemplo presentado. En ella se muestra como la columna "discount" perteneciente a la tabla "Product" ha sido eliminada, perdiendo así todos sus datos.

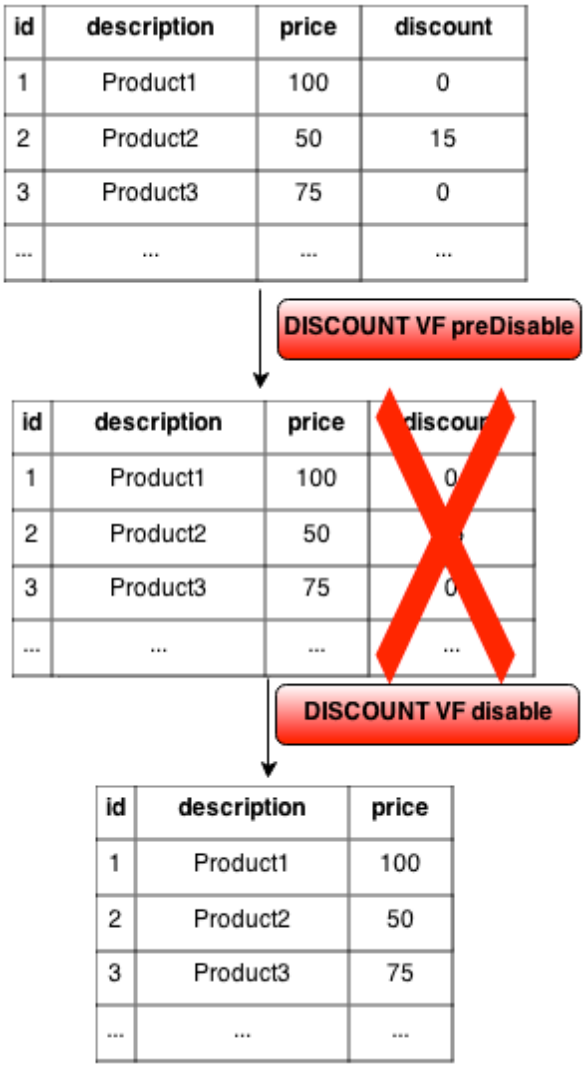


Ilustración 26: Eliminación de columna

- **Actualización de columna:** La estrategia de actualización de columna garantiza el correcto manejo de una funcionalidad volátil, siempre y cuando no sean de importancia los datos introducidos por el nuevo atributo. Consta de mantener el

mapeo de la columna dada y además actualizar todos sus datos modificados con valores significantes (es decir distintos al valor por defecto) a un valor por defecto, para que la próxima vez que la funcionalidad volátil sea activada, estos no conserven sus valores anteriores. En realidad, la necesidad de utilizar esta estrategia está relacionada en gran parte a la manera en que se inicialicen los valores en las etapas de "inicialización de activación" de las distintas funcionalidades. Igualmente, la ejecución de esta actualización brinda un mayor marco de seguridad para la gestión del VF Framework.
Para esta estrategia resulta fundamental el hecho de que la columna incorporada en la etapa de activación esté configurada como NULLABLE o posea un valor por defecto. Así, cuando la funcionalidad volátil sea desactivada, se podrán seguir persistiendo instancias de la entidad en cuestión sin ningún tipo de problema.
Esta estrategia ha sido aplicada y puede observarse en la Ilustración 27 correspondiente al dominio del ejemplo presentado. En ella se muestra como todos los valores de la columna "discount" perteneciente a la tabla "Product", han sido actualizados al valor por defecto "0".

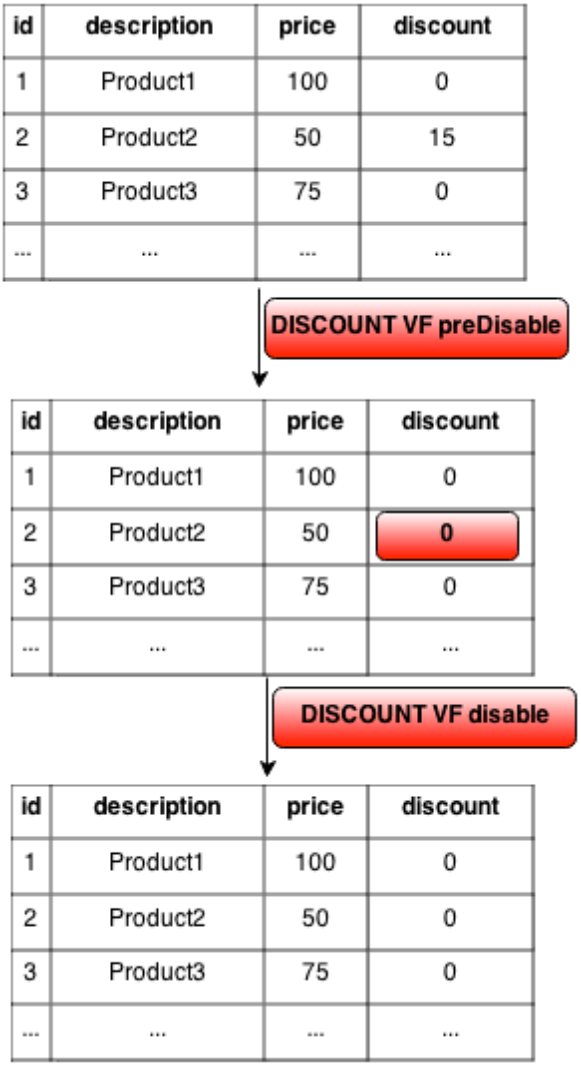


Ilustración 27: Actualización de columna

- **Backup de la tabla:** Tanto la estrategia de eliminación de columna como la estrategia de actualización de columna pueden ser combinadas con el resguardo de backups de la tabla de la base datos. De esta manera, no habría pérdida de información ya que la misma podría ser recuperada en el futuro de ser necesaria. Además, se obtendría un registro histórico mediante el cual podrían observarse las modificaciones que han sufrido los datos de determinada entidad. Esta opción es válida, aunque también podría traer ciertas desventajas. En caso de precisarse múltiples backups, esta solución podría volverse poco escalable, complejizándose en gran medida su tratamiento. Ello dependerá mucho de la situación en cuestión y de la manera en que los backups sean realizados. Su utilización tiene sentido principalmente para entidades con datos dinámicos, cuyas modificaciones de valores a lo largo del tiempo deban quedar registradas.
  Teniendo en cuenta que la tabla "Product" del ejemplo dado es una entidad con datos dinámicos, esta estrategia se ajustaría perfectamente a dicho caso. Esto es así dado que como el descuento establecido para los productos puede variar a lo largo del tiempo, ésta sería una manera de guardar su registro. Como puede observarse en la Ilustración 28, la "PreDesactivación" incluye la realización de un backup de la tabla "Product" y la posterior eliminación de la columna "discount".

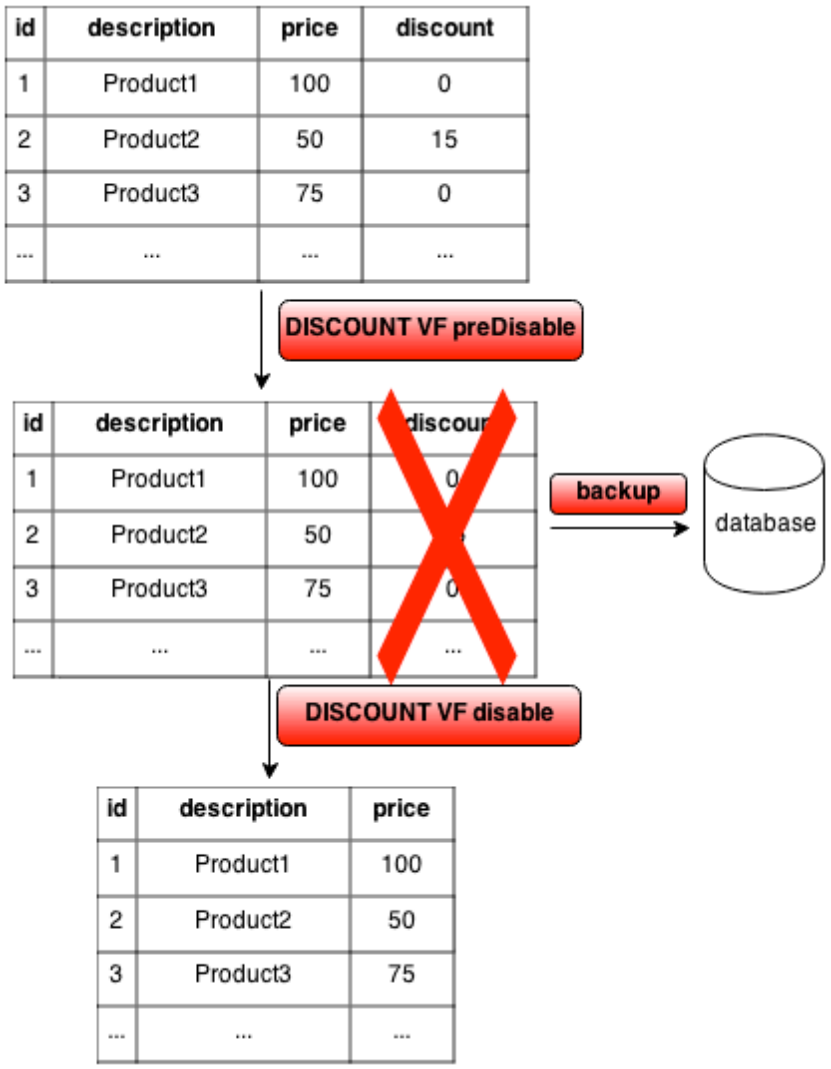


Ilustración 28: Backup de la tabla

- **Conservación de datos y columna:** Esta es la estrategia que mejor se ajusta a aquellas ocasiones en las que se cuenta con entidades con datos estáticos de importante relevancia. Con ella, no debe efectuarse ningún tipo de cambio en las tablas de la base de datos. Los datos contenidos deben ser resguardados, pero como no interfieren con el funcionamiento normal de la aplicación, no requieren ningún tipo de acción extra.
  Esta estrategia es ideal para casos como el de los registros de la clase SaleItem que componen una venta en el ejemplo dado. Esto puede ser observado en la Ilustración 29.

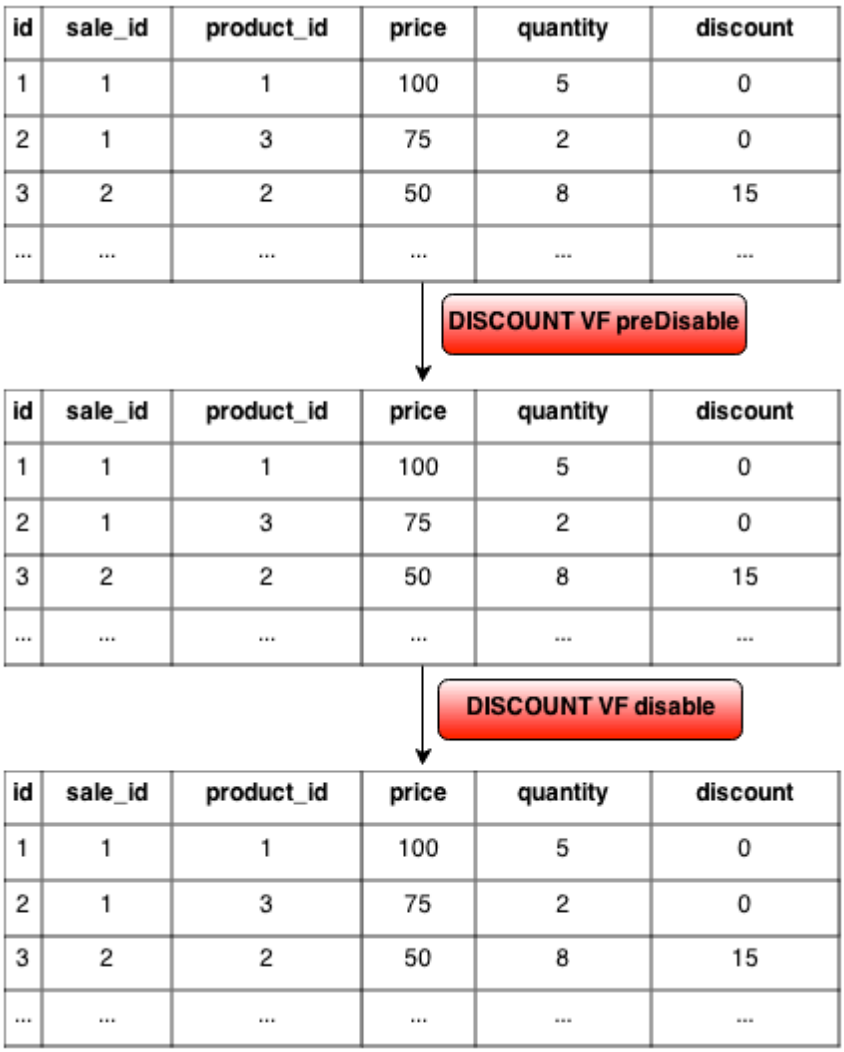


Ilustración 29: Conservación de datos y columna

5.4.3.3.3 Recomendaciones extra de uso

Si en lugar de involucrar un atributo, una funcionalidad volátil involucrara la introducción de una relación entre entidades, podrían utilizarse los mismos tipos de estrategias. Por supuesto, también habría que evaluar el tipo de mapeo en cuestión para cada caso en particular.

Se ha puesto especial énfasis en el tratamiento de los datos debido a que es un tema de gran relevancia, que podría ser tratado satisfactoriamente mediante las dos etapas que han sido definidas. Sin embargo, tanto la “Inicialización de activación” como

la “PreDesactivación” no son más que dos etapas configurables que están presentes en los procesos de activación y desactivación de las funcionalidades volátiles del VF Framework. Como tales, podrían ser empleadas para lanzar cualquier tipo de acción que crea conveniente el usuario de este Framework.

Además, se debe tener en cuenta que no todas las funcionalidades volátiles requieren de la definición de estas subetapas obligatoriamente. Esta cuestión depende y debe ser analizada según el caso particular. En ciertos casos, inclusive el valor nulo puede tener un valor significativo diferente al 0, que no debería ser modificado.

Tomemos el caso de la funcionalidad volátil “GIFT VF” del ejemplo. Esta funcionalidad incorpora el atributo “gift” a la entidad Sale. Este atributo es un dato de tipo booleano que se setea al realizarse una venta. Cuando la funcionalidad volátil se encuentra activada, si el monto de la misma supera los $1000, se le entrega un regalo al cliente y el atributo “gift” queda seteado en true. De lo contrario, no se entrega ningún regalo y queda seteado en false. Cuando la funcionalidad volátil se encuentra desactivada, el atributo “gift” queda seteado en NULL, sin importar el monto de la venta. Estos tres valores representan tres situaciones diferentes que podría ser importante diferenciar y conservar. Si sólo se considerara la posibilidad de contener valores true y false, se estaría perdiendo información.

En la Ilustración 30 se muestra la tabla de verdad que representa las condiciones y acciones a tomar para el ejemplo recién dado.

Condiciones			
GIFT VF	V	V	F
Total > $1.000	V	F	-
Acciones			
Setear campo “gift” en True	X		
Setear campo “gift” en False		X	
Setear campo “gift” en Null			X

Ilustración 30: Tabla de verdad de tratamiento de atributo

5.4.3.4 Commons para las funcionalidades volátiles

Múltiples funcionalidades volátiles pueden tener un comportamiento común e involucrar cambios coincidentes en una aplicación web. Esto podría desembocar en una superposición de las mismas, lo cual tendría un efecto contraproducente para el correcto funcionamiento de la aplicación. Por esta razón, surgió la idea de desacoplar aún más las funcionalidades volátiles, brindando la posibilidad de incluir otras funcionalidades como “Commons”. De esta manera, se logra además una mayor reutilización, una mayor modularización y una simplificación del mantenimiento durante el ciclo de vida de las funcionalidades volátiles.

Las funcionalidades volátiles de VF Framework pueden ser incluidas dentro de otras funcionalidades volátiles. Esta característica permite la reutilización de las modificaciones definidas en ellas simplemente mediante su inclusión. Además, por supuesto, una funcionalidad volátil que contenga otras funcionalidades también podrá agregar nuevas modificaciones propias.

Para que este asunto pueda ser cubierto, resulta necesaria la introducción de una relación bidireccional en la entidad que representa las funcionalidades volátiles. Cada funcionalidad volátil debe tener conocimiento por un lado de aquellas funcionalidades incluidas como commons (includedVFs,) y por otro lado de aquellas funcionalidades

dependientes (depdendentVFs), es decir aquellas que las han agregado como "Common". Esta relación bidireccional ha sido representada en el diagrama UML de la Ilustración 31.

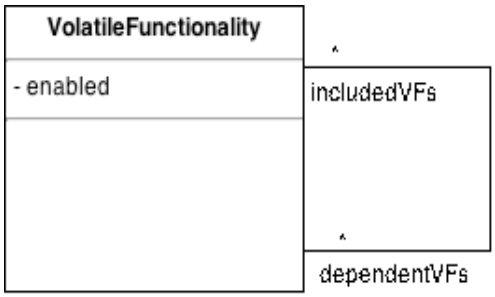


Ilustración 31: Commons para las FVs

Cuando múltiples funcionalidades volátiles comparten una funcionalidad volátil como "Common", aparecen ciertas cuestiones que deben ser tenidas en cuenta a la hora de gestionarlas. Estas cuestiones se relacionan con todas las etapas del ciclo de vida de las funcionalidades volátiles, las cuales también incluyen sus subetapas. A continuación se presentan las mismas:

- **Activación:** Para poder activar una funcionalidad volátil que contiene otras funcionalidades volátiles, deben activarse en primera instancia todas aquellas funcionalidades incluidas que se encuentren desactivadas. Una vez finalizada esta tarea, se podrá activar la funcionalidad principal. Este mecanismo previene la superposición de funcionalidades con comportamiento similar. Si varias funcionalidades comparten una funcionalidad incluida, bastará con activar esta última una sola vez.
- **Inicialización de activación:** La inicialización de activación continúa funcionando de la misma manera. La única cuestión a la que hay que prestarle atención es a asignarle el valor por defecto sólo a aquellos datos que posean la columna del atributo incorporado con valor nulo. Luego, sólo deberían modificarse los datos a los que se les quiere asignar un valor significativo propio de la funcionalidad volátil. El resto de los datos no deberían ser alterados, de lo contrario los valores asignados por otra funcionalidad podrían perderse.
- **PreDesactivación:** Si la funcionalidad volátil involucrara modificaciones en una entidad con datos estáticos la mejor estrategia seguiría siendo la de conservación de datos y columna. En cambio, si involucrara una entidad con datos dinámicos, la estrategia de eliminación de columna pasaría a ser obsoleta. Su utilización produciría la eliminación de una columna, mientras que otra funcionalidad podría precisar de ella. Por esta razón, la mejor y única opción pasaría a ser la utilización de la estrategia de actualización de columna.
- **Desactivación:** Para poder desactivar una funcionalidad volátil, esta no debe estar incluida dentro de otra funcionalidad activada. Si no lo está, entonces la funcionalidad volátil podrá ser desactivada sin problemas. Además, a su vez si la funcionalidad en cuestión incluye otras funcionalidades volátiles, deberán desactivarse sólo aquellas que no estén contenidas dentro de otras funcionalidades activadas.

Una funcionalidad volátil podría ser utilizada como "Common", por ejemplo, si existieran múltiples promociones de descuentos independientes entre sí. Supongamos

que una aplicación web requiera definir una promoción de 30% de descuento para las ventas del producto "Producto1" y una promoción de 20% de descuento para las ventas de los productos con precio mayor a $1.000.

En ese caso, sería recomendable:

- Crear la funcionalidad volátil genérica "DISCOUNT VF". Esta funcionalidad solamente definiría cambios comunes a cualquier funcionalidad cuyo objetivo fuera la definición de una promoción con descuentos. Estos cambios incluirían cambios tales como la inyección y el mapeo de una nueva variable para almacenar el descuento, la visualización del descuento en determinadas vistas y el cálculo del precio final considerando el descuento.
- Crear la funcionalidad volátil "PRODUCT1 DISCOUNT VF". Esta funcionalidad incluiría a "DISCOUNT VF" como "Common". Además, definiría cuestiones particulares como la inicialización del descuento en 30% para el producto "Product1" en la etapa de "Inicialización de activación" y la incorporación de banners en la vistas.
- Crear la funcionalidad volátil "THOUSAND DISCOUNT VF". Esta funcionalidad incluiría a "DISCOUNT VF" como "Common". Además, definiría cuestiones particulares como la inicialización del descuento en 20% para los productos con precio mayor a $1.000 en la etapa de "Inicialización de activación" y la incorporación de banners en la vistas.

5.4.3.5 Programación de eventos de activación y desactivación

Como ya se ha destacado, VF Framework propone una solución dinámica, en la que la activación y la desactivación de las funcionalidades volátiles pueden programarse a partir de eventos. El resultado de la ejecución de estos eventos debe poder ser visualizado en tiempo de ejecución.

Existen tres tipos de eventos:

- **Eventos temporales:** Eventos configurados para que ser ejecutados en un momento dado. Permiten la programación de activaciones y desactivaciones cada cierta cantidad de tiempo o para un momento particular. Pueden definirse tanto para ser ejecutados una única vez, como también para ser ejecutados con determinada periodicidad. En el ejemplo dado, mediante un evento temporal podría configurarse la activación de la funcionalidad volátil "GIFT VF" para el día 17 de Diciembre a las 0hs.
- **Eventos de negocio:** Eventos configurados para ser ejecutados en una situación dada. Permiten la programación de activaciones y desactivaciones ante la aparición de situaciones particulares relacionadas con la lógica de negocio de una aplicación.
  En el ejemplo dado, mediante un evento de negocio, podría configurarse la desactivación de la funcionalidad volátil "GIFT VF" para el momento en que se acaben el stock de los regalos.
- **Eventos de negocio temporales:** Eventos configurados para ser ejecutados en un momento dado y en una situación dada. Son una combinación de eventos temporales y de eventos de negocio.
  En el ejemplo dado, mediante un evento de negocio temporal, podría configurarse la activación de la funcionalidad volátil "DISCOUNT VF" para el momento en que el stock del producto "Product1" sea inferior a

50 unidades, contando a partir del día 1 de Marzo. Así podría programarse una liquidación de productos.

El lenguaje de programación elegido debe contar con un mecanismo que permita el establecimiento de este tipo de eventos. Para ello se destacan tres posibles implementaciones:

- **Motor de reglas:** La utilización de un motor de reglas resulta una solución muy útil y natural que permite la definición de eventos de activación y desactivación de manera sencilla y eficiente. Simplemente requiere la definición de un archivo con las reglas de activación y desactivación. Cada regla puede contar con especificaciones de tiempo y/o cuestiones particulares referidas a las reglas de negocio de la aplicación. Una vez alcanzadas esas condiciones definidas se lanzarán las acciones a ejecutar (activaciones y/o desactivaciones). Esta opción permite la definición de los tres tipos de eventos y cumple perfectamente con el objetivo deseado.
- **AOP + Scheduler:** La utilización de programación orientada a aspectos y un Scheduler es una buena opción especialmente ante la ausencia de un motor de reglas. La combinación de estas dos herramientas permite el reemplazo de éste último, obteniéndose así resultados finales similares. Los eventos pueden ser definidos mediante el Scheduler; los eventos de negocio pueden ser definidos mediante aspectos; y los eventos de negocio temporales pueden ser definidos mediante una combinación del Scheduler y aspectos.
  La programación de este último tipo de eventos variará según se cuente con un lenguaje estáticamente o dinámicamente tipado:
  - **Lenguaje de programación dinámicamente tipado:** Utilizando un lenguaje dinámicamente tipado, en primer lugar, se debe definir un evento temporal mediante el Scheduler. Al cumplirse dicho evento temporal, se debe agregar un evento de negocio (con un aspecto). Recién cuando se cumpla la situación especificada por este evento de negocio, se dará lugar a la activación o desactivación programada para la funcionalidad volátil. Por último, el evento de negocio deberá ser desactivado.
    En la Ilustración 32, puede observarse la red de Petri que representa a este proceso para eventos de negocio temporales de activación en lenguajes dinámicamente tipados. Por otro lado, en la Ilustración 33, puede observarse un diagrama similar para eventos de negocio temporales de desactivación.

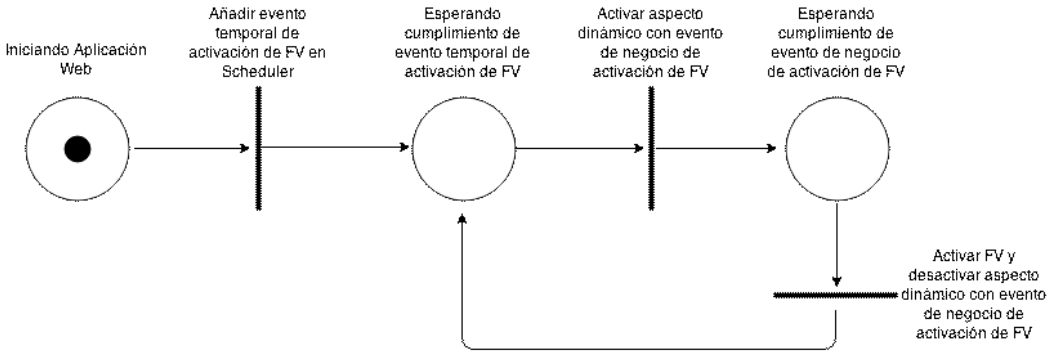


Ilustración 32: Activación con evento de negocio temporal en lenguaje dinámico

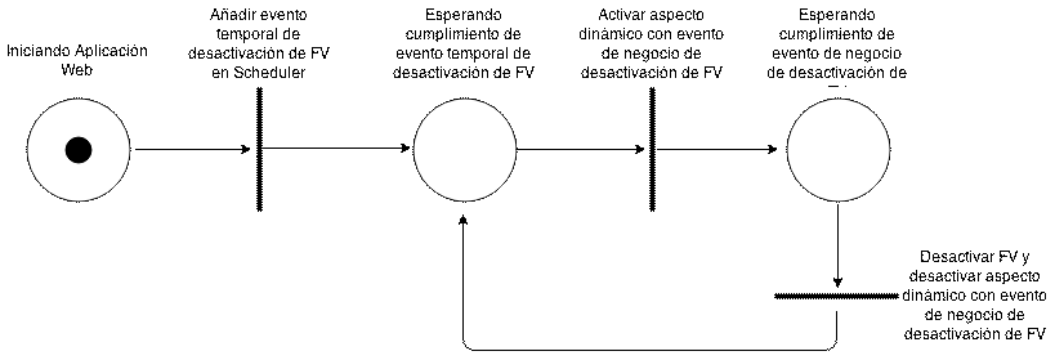


Ilustración 33: Desactivación con evento de negocio temporal en lenguaje dinámico

- **Lenguaje de programación estáticamente tipado:** Utilizando un lenguaje estáticamente tipado, debe definirse un evento temporal mediante el Scheduler, pero además es necesario el almacenamiento del estado del mismo. El valor de este evento debe ser activado al cumplirse con las condiciones temporales de activación o desactivación especificadas. Por otra parte, se define un evento de negocio (con un aspecto estático), cuya ejecución depende tanto del estado del evento temporal, como de alguna situación referida a la lógica de negocio. Luego de cumplirse ambas condiciones y ejecutarse la activación o desactivación configurada, el valor del evento temporal debe ser desactivado nuevamente.
  En la Ilustración 34, puede observarse la red de Petri que representa a este proceso para eventos de negocio temporales de activación para lenguajes estáticamente tipados. Por otro lado, en la Ilustración 35, puede observarse un diagrama similar para eventos de negocio temporales de desactivación.

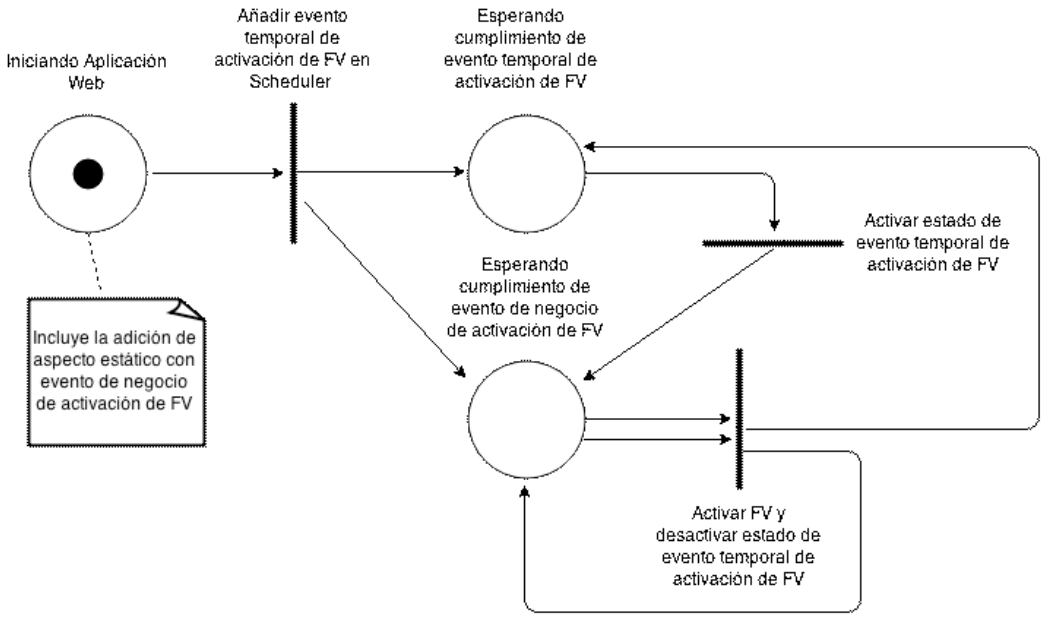


Ilustración 34: Activación con evento de negocio temporal en lenguaje estático

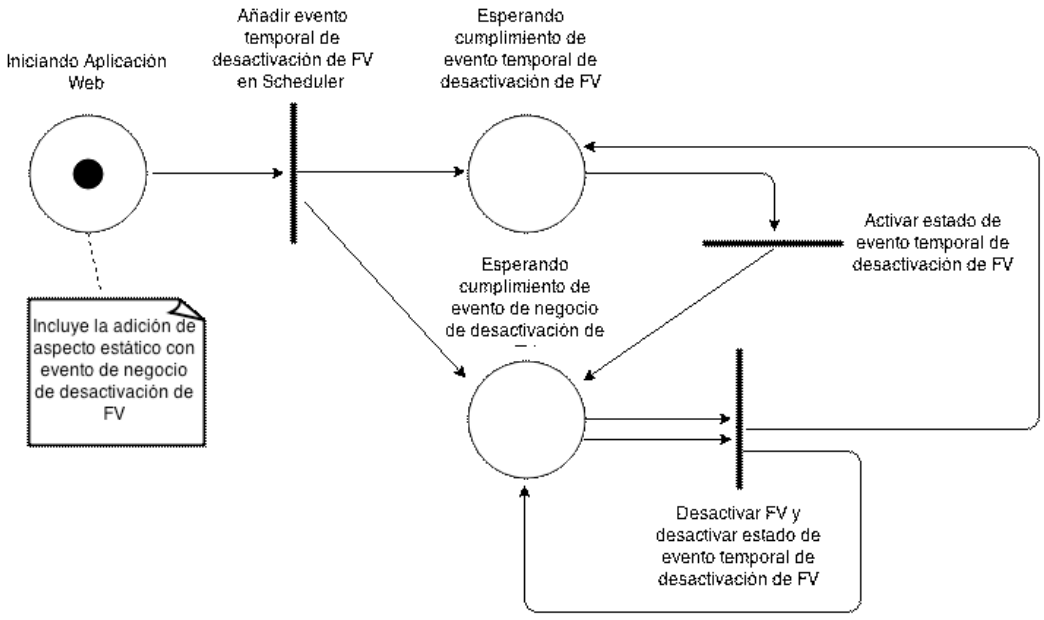


Ilustración 35: Desactivación con evento de negocio temporal en lenguaje estático

6 Implementación en Java

6.1 Objetivo

Con el objetivo de probar el marco de trabajo presentado, se implementó un prototipo para el lenguaje de programación Java. La implementación del mismo tiene como fin no sólo una demostración práctica y concreta del mismo, sino también la descripción de los pasos a seguir para lograr incorporar funcionalidades volátiles en aplicaciones web desarrolladas con este lenguaje tan utilizado.

6.2 Tecnologías utilizadas

Para implementar funcionalidades volátiles siguiendo el VF Framework para el lenguaje de programación Java, se han utilizado las siguientes tecnologías:

- Java 1.7
- Spring 3
- Spring MVC 3
- AspectJ
- Saxon XSLT 2.0
- Apache Maven 3.1.0
- Drools 5.4.0

6.3 Implementación

Para cumplir con el objetivo deseado de incorporar funcionalidades volátiles como VF Framework propone, se decidió:

- Mantener sin alterar la aplicación original
- Implementar un nuevo proyecto con las funcionalidades volátiles que decoran a la aplicación original
- Utilizar AOP (en lenguaje estáticamente tipado)
- Utilizar transformaciones XSL
- Utilizar weaving estático
- Utilizar un motor de reglas

Mediante este tipo de implementación resulta una tarea muy sencilla el proceso de compilar y correr la aplicación original con o sin las funcionalidades volátiles añadidas. Si se opta por correr la aplicación original sin cambios, no será más que compilar y correr la aplicación de la manera tradicional. Si en cambio se optara por correr la aplicación con las nuevas funcionalidades volátiles añadidas, se deberá compilar la aplicación original, para luego correr el nuevo proyecto (que contiene todos los cambios para incorporar las nuevas funcionalidades). Este nuevo proyecto será el encargado de construir el nuevo archivo WAR que será ejecutado.

Las funcionalidades volátiles se adicionan definiendo principalmente un conjunto de aspectos y de transformaciones XSL que se aplican utilizando Maven sobre la aplicación original compilada. De esta manera, las funcionalidades volátiles no son intrusivas con la aplicación original y pueden ser agregadas dentro del ciclo de compilación.

Otra característica importante que posee el enfoque aquí presentado es la posibilidad de programar la activación y la desactivación de las funcionalidades volátiles, lo cual se logra mediante el motor de reglas Drools.

6.4 Prototipo desarrollado

6.4.1 Descripción

Para el desarrollo del prototipo se tomó como base a la aplicación web booking-faces [12] (sample provisto por Spring Web Flow) y se le incorporaron algunas funcionalidades volátiles.

Booking-faces es una aplicación que permite gestionar la reserva de habitaciones para hoteles. Ya sea mediante el listado de hoteles o mediante la búsqueda de uno específico, un usuario registrado puede reservar un cuarto. En el proceso de reserva se solicitan datos tales como nombre, número de teléfono, e-mail, e información de la tarjeta de crédito. La aplicación fue desarrollada utilizando el lenguaje Java y frameworks de código abierto como SpringMVC, JSF/PrimeFaces y Hibernate, entre otros.

Para esta aplicación se ha decidido introducir las siguientes funcionalidades volátiles:

- **Promoción del fin de semana largo (LongWeekenPromotionVF):** El gobierno introduce al calendario laboral feriados para fomentar el consumo interno. En este caso, los hoteles que pertenecen al país con la promoción pueden promover sus habitaciones con ofertas del tipo "tomen 3 noches y paguen 2". Esta promoción es válida para reservas que incluyan el período del 21/03/2015 al 24/03/2015. Dentro del conjunto de cambios introducidos por este requisito se encuentran:
  - Banner de la promoción en la cabecera de todas las páginas. El mismo se encuentra resaltado con un rectángulo color rojo en la Ilustración 36.

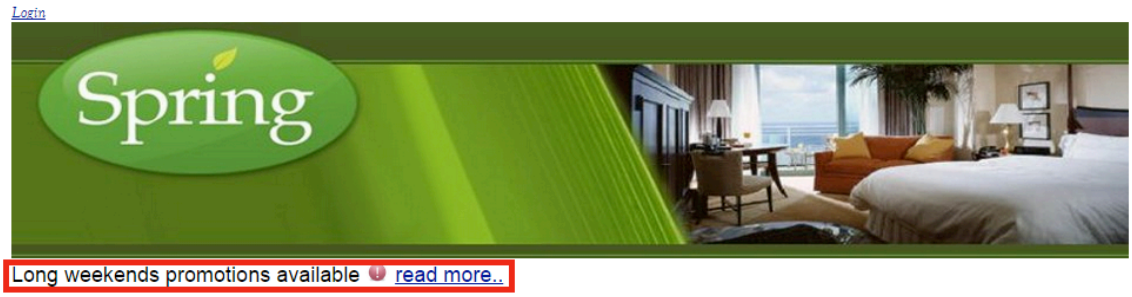


Ilustración 36: Prototipo Java - LongWeekenPromotionVF - Banner

- Nueva página con detalles de la promoción, la cual contiene un mensaje que se encuentra resaltada con un rectángulo color rojo en la Ilustración 37.

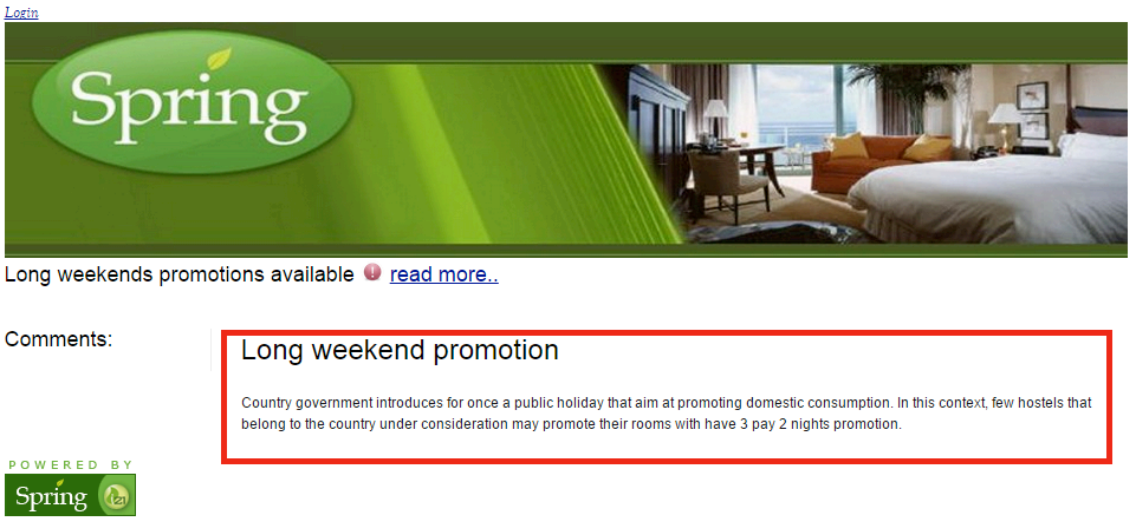


Ilustración 37: Prototipo Java - LongWeekenPromotionVF - Página

- Formulario de carga de reserva enriquecido. Se informa, a medida que se introducen los datos de la reserva, si la reserva posee promoción en el caso de que el período de la reserva incluya el fin de semana largo del 21/03/2015 (check in) al 24/03/2015 (check out). Cuando esto sucede aparece un pop-up como el que se ve en la Ilustración 38 resaltado con un rectángulo color rojo.

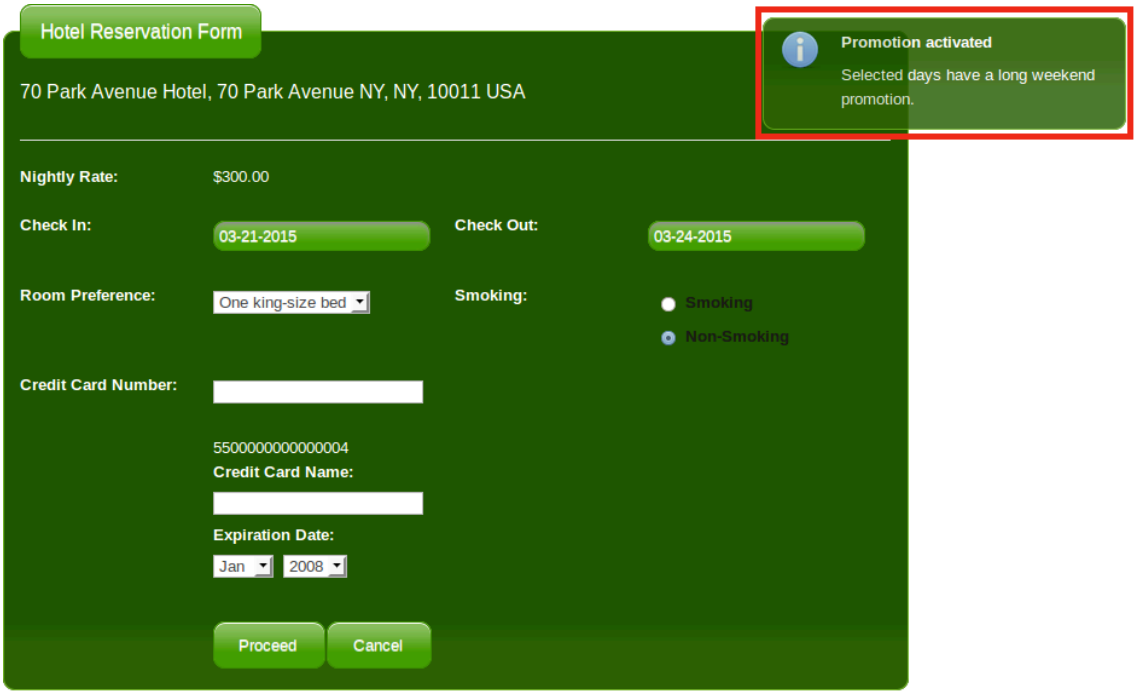


Ilustración 38: Prototipo Java - LongWeekenPromotionVF - Formulario enriquecido

- El monto total de la reserva contempla la aplicación de la promoción y se informa la aplicación de la misma. Esto puede observarse en la Ilustración 39, a donde se encuentran resaltados con rectángulos color rojo el monto total a pagar (el precio por noche del Hotel 70 Park Avenue es de $300 y se está cobrando $600 por tres noches) y el mensaje que dice que la reserva que se está realizando aplica para la promoción.

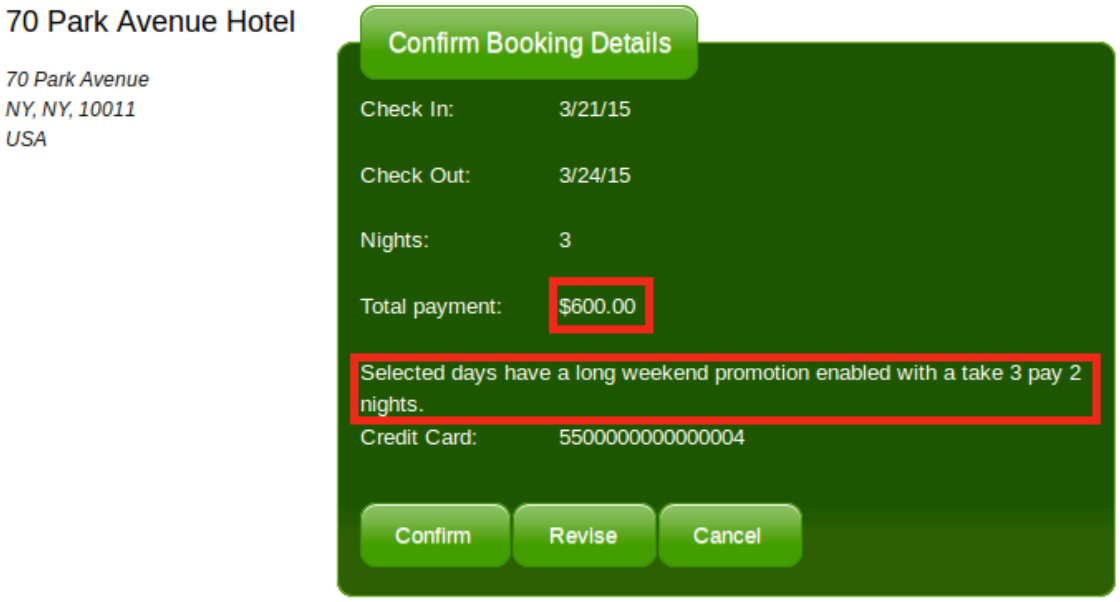


Ilustración 39: Prototipo Java - LongWeekenPromotionVF - Monto total

- Como puede observarse en la Ilustración 40 resaltado con un rectángulo color rojo, el listado de reservas ha sido enriquecido. Se agregaron dos columnas, una con el monto de la reserva y otro para indicar si la misma incluye promoción mediante un flag.

Ilustración 40: Prototipo Java - LongWeekenPromotionVF - Listado enriquecido

- **Auditoria de consumo (NotificationVF):** Como política de control fiscal, un gobierno puede disponer una ley de consumo donde cualquier prestación de servicio superior a un límite debe ser informado. Es decir, si el hotel brinda servicios (alquiler de habitaciones) por un monto superior a los $2000, éste debe notificar al estado quién ha realizado la compra.
  Dentro del conjunto de cambios introducidos por este requisito se encuentran:
  - Simulación de notificación enviada al estado mediante impresión de mensaje en la consola. En la Ilustración 41, puede visualizarse resaltada con un rectángulo color rojo, una simulación de estas.

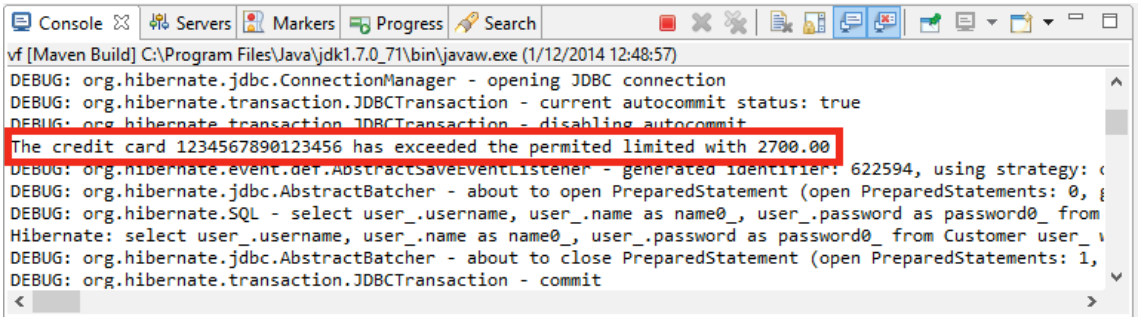


Ilustración 41: Prototipo Java - NotificationVF – Envío de notificación

Desarrollo

En una primera instancia, se han incorporado todas estas funcionalidades volátiles a la aplicación de la manera tradicional o ad-hoc, es decir, añadiendo código disperso por toda la aplicación.

Luego, siguiendo los conceptos definidos por VF Framework, se mantuvo el proyecto original (proyecto llamado "booking-faces") y se desarrolló un nuevo proyecto desacoplado (proyecto llamado "vf"). En este último proyecto se definieron todos los cambios a efectuar para poder enriquecer la aplicación original. De esta manera, fue posible lograr un funcionamiento semejante al de una aplicación ad-hoc, sumando además las ventajas del Framework presentado.

Nota: Para poder visualizar la activación y la desactivación de las funcionalidades volátiles del prototipo se programaron las reglas para que estas sean activadas y desactivadas periódicamente. Al ejecutar la aplicación con las funcionalidades volátiles, la promoción del fin de semana largo se encontrará activada y la de auditoría de consumo desactivada. Cada 5 minutos sus estados se irán alternando.

6.4.2 Ejecución

En el Apéndice A, se describe cómo llevar a cabo la instalación del prototipo desarrollado.

Como será explicado a lo largo de la guía, una vez instalado el prototipo, podemos optar entre ejecutarlo con o sin funcionalidades volátiles.

A continuación se detallan los pasos a seguir para ambas ejecuciones:

6.4.2.1 Ejecución de la aplicación original

1) Acceder a la perspectiva de Spring (Menú Windows > Open Perspective >Other, Spring).
2) Seleccionar el proyecto original, hacer click derecho sobre el proyecto original, seleccionar “configure” y luego “Convert to maven Project”.
3) Seleccionar el proyecto original, hacer click derecho y seleccionar Run as > maven install.
4) Seleccionar el proyecto original, hacer click derecho y luego seleccionar Run as > maven build.. , ingresar “tomcat:run” en el campo goals, y por último presionar el botón “Run”.
5) Abrir la URL http://localhost:8080/booking-faces con un navegador.

6.4.2.2 Ejecución de la aplicación con las funcionalidades volátiles

1) Acceder a la perspectiva de Spring (Menú Windows > Open Perspective
2) >Other, Spring).
3) Seleccionar el proyecto original, hacer click derecho sobre el proyecto original, seleccionar “configure” y luego “Convert to maven Project”.
4) Seleccionar el proyecto original, hacer click derecho y seleccionar Run as > maven install.
5) Seleccionar el proyecto vf, hacer click derecho sobre el proyecto original, seleccionar “configure” y luego “Convert to maven Project”.
6) Seleccionar el proyecto vf, hacer click derecho y seleccionar Run as > maven install.
7) Seleccionar el proyecto vf, hacer click derecho y luego seleccionar Run as > maven build.. , ingresar “clean jetty:run-war” en el campo goals, y por último presionar el botón “Run”.
8) Abrir la URL http://localhost:8080/booking-faces con un navegador.

6.5 Guía para la incorporación de funcionalidades volátiles

6.5.1 Objetivo

El objetivo de esta guía es la especificación de todos los pasos a seguir para lograr la incorporación de funcionalidades volátiles en una aplicación web desarrollada en Java, siguiendo los conceptos de VF Framework.

6.5.2 Arquitectura desacoplada

Para desacoplar la arquitectura correspondiente a las funcionalidades volátiles, lo primero que se deberá hacer será crear un nuevo proyecto Java, que en la guía será llamado "vf". Este proyecto contendrá todo el código necesario para implementar las nuevas funcionalidades e integrarlas al proyecto base (el cual a su vez permanecerá inalterado).

El proyecto creado deberá ser convertido en un proyecto Maven, ya que se utilizará el archivo pom.xml (que este tipo de proyectos contienen) para bajar todas las dependencias necesarias y para definir la construcción de la aplicación con las nuevas funcionalidades volátiles.

Cuando se convierta el proyecto “vf” a Maven, se deberá ingresar el groupId, el artifactId y el packaging. El groupId a ingresar es indistinto, por lo cual podrá elegir cualquiera que desee. En cambio, el packaging a ingresar debe ser “war” y el artifactId debe ser el mismo que posee la aplicación original. Es importante que el artifactId ingresado sea exactamente igual al de la aplicación original, de lo contrario cuando la aplicación sea compilada con funcionalidades volátiles, la url de la misma cambiará. Si por ejemplo se definiera “vf” como artifactId, entonces la url de la aplicación pasaría a ser “http://localhost:8080//vf”.

Una vez que el proyecto “vf” haya sido convertido a Maven, el archivo pom.xml será añadido automáticamente. En él se deberá definir como dependencia a la aplicación web original. Es indispensable definir classifier como “classes” para que pueda ser referenciada correctamente. La dependencia a agregar quedaría de la siguiente manera:

```
<dependency>
        <groupId>org.springframework.webflow.samples</groupId>
        <artifactId>app-web</artifactId>
        <classifier>classes</classifier>
        <version>1.0.0.BUILD-SNAPSHOT</version>
        <scope>provided</scope>
        <exclusions>
                <exclusion>
                <artifactId>spring-context</artifactId>
                <groupId>org.springframework</groupId>
                </exclusion>
                <exclusion>
                <artifactId>spring-core</artifactId>
                <groupId>org.springframework</groupId>
                </exclusion>
                <exclusion>
                <artifactId>spring-beans</artifactId>
                <groupId>org.springframework</groupId>
                </exclusion>
        </exclusions>
</dependency>
```

Ya se ha dejado bien en claro la importancia de mantener de manera inalterada la aplicación original. Sin embargo, para que el proyecto “vf” pueda decorar el código fuente de las clases pertenecientes a la aplicación original, resulta inevitable efectuar un pequeño y único cambio en el archivo pom.xml de la aplicación original. Este pequeño cambio será la adición del siguiente plugin:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
         <attachClasses>true</attachClasses>
</configuration>
```

```
</plugin>
```

Una vez guardados ambos archivos pom.xml, se deberá ejecutar el comando "mvn clean install" primero para el proyecto original y luego para el proyecto "vf", y así efectivamente todas las dependencias serán descargadas para ambos.

El siguiente paso será crear la carpeta "volatile" dentro de la carpeta src del proyecto "vf".

A su vez, la carpeta "volatile" deberá contener 3 nuevas subcarpetas:

- **java:** Contendrá el paquete en el que se ubicarán todas las nuevas clases Java.
- **aspects:** Contendrá el paquete en el que se ubicarán todos los aspectos.
- **resources:** Contendrá los paquetes para cada uno de los diferentes tipos de recursos nuevos (config, images, properties, rules, styles, xhtmls, xsls) y se ubicarán los recursos en los paquetes según correspondan.

Es importante configurar estos 3 directorios como source folders durante el desarrollo de la nueva funcionalidad volátil. De esta manera, se logrará visualizar errores en compilación y se asegura que todos los archivos sean encontrados.

En la Ilustración 42 se muestra cómo debería quedar a estructura resultante del proyecto "vf".

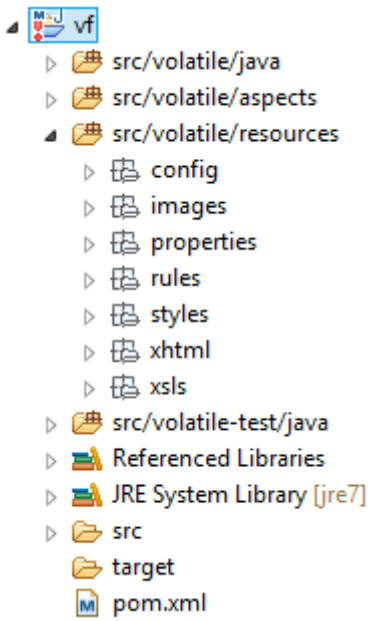


Ilustración 42: Estructura del proyecto "vf"

Nota: Lógicamente sería posible y tendría sentido que existieran múltiples proyectos desacoplados (uno por cada funcionalidad volátil) y que los mismos pudieran ser encadenados con el propósito de correr una aplicación con las funcionalidades volátiles deseadas. Sin embargo, por cuestiones de tiempo, para este prototipo no se han desacoplado las funcionalidades volátiles unas de otras, sino que estarán todas contenidas en el mismo proyecto.

6.5.3 Modificación de las capas de la aplicación

Como ya ha sido explicado, la introducción de funcionalidades volátiles en una aplicación web involucra la modificación de cada una de sus capas. A continuación se detallará el uso de las tecnologías y estrategias utilizadas para lograr dicho objetivo en este lenguaje de programación.

6.5.3.1 Modificación de la capa de negocio

Para modificar la capa de negocio de manera de no alterar el código fuente de la aplicación original se utilizan aspectos. Si lo que se desea es modificar la lógica de negocios de la aplicación original, la utilización del framework AspectJ resulta muy interesante. Este framework permite introducir modificaciones en el comportamiento de una aplicación, sin necesidad de modificar su código fuente original. Simplemente mediante la definición de aspectos es posible modificar de manera desacoplada la estructura y el comportamiento de clases Java. Dado que Java es un lenguaje de programación estáticamente tipado, estas modificaciones se incorporan en tiempo de compilación.

Para poder utilizar AspectJ es necesario descargar sus dependencias. Para ello se debe agregar la siguiente dependencia al archivo pom.xml de la aplicación "vf".

```
<!--AspectJ  -->
        <dependency>
                <groupId>org.aspectj</groupId>
                <artifactId>aspectjrt</artifactId>
                <version>1.6.7</version>
</dependency>
```

Luego de agregar esta dependencia, se deberán actualizar nuevamente las dependencias del proyecto "vf" ejecutando el comando "mvn clean install".

Será necesario crear un aspecto para cada una de las funcionalidades volátiles a incorporar. Dentro de cada uno de los aspectos, se definirán las modificaciones a efectuar para todas las clases afectadas por la funcionalidad volátil en cuestión. Estos aspectos deben ser ubicados dentro de la carpeta src/volatile/aspects.

Supongamos que su aplicación requiere la incorporación de una funcionalidad volátil para el lanzamiento de descuentos (DiscountVF). En tal caso, el primer paso será definir un aspecto como el siguiente:

```
public aspect DiscountVFAspect {

}
```

Inyección de variables y relaciones

En caso de querer inyectarle un nuevo atributo o relación a una clase original afectada por una funcionalidad volátil, puede realizarlo dentro del aspecto definido de la siguiente manera:

```
private BigDecimal Product.discount = 0;
```

Mapeo de variables y relaciones

El mapeo de variables y relaciones es una cuestión que puede ser resuelta de manera muy sencilla. Si la aplicación cuenta con persistencia (por ejemplo utilizando JPA) y define los mapeos a partir de anotaciones, los nuevos atributos y relaciones de una clase serán mapeados de igual manera que lo son sus atributos y relaciones originales. Consecuentemente, también pueden definirse anotaciones para estos nuevos mapeos, que servirán de metadata para el framework de persistencia.

```
@NotNull
private BigDecimal Product.discount = 0;
```

Nota: Si los mapeos no son definidos a partir de anotaciones, se deberían definir transformaciones XSL en las que los nuevos mapeos sean incorporados a los archivos de configuración de mapeos correspondientes.

Inyección de métodos

Las modificaciones efectuadas en la estructura de una clase suelen requerir también modificaciones en el comportamiento de la misma. Así como pueden inyectarse nuevos atributos en una clase, también pueden inyectarse nuevos métodos.

A continuación se muestra como pueden inyectarse getters y setters para un nuevo atributo, así también como un nuevo método:

```
public BigDecimal Product.getDiscount() {
        return discount;
    }

public void Product.setDiscount(BigDecimal discount){
        this.discount = discount;
}

public boolean Product.hasDiscount(){
        return (this.getDiscount == 0);
}
```

Modificación del flujo original de métodos existentes

En Java, la modificación del flujo original de métodos existentes puede lograrse a partir de la intercepción de métodos. Este mecanismo, propio de la programación orientada a aspectos, permite capturar llamadas a métodos y modificar su comportamiento resultante mediante la definición de pointcuts y advices asociados.

La clase "Purchase" será tomada para ejemplificar el uso de este punto.

Para llevar a cabo la definición de un pointcut es necesario definir un nombre y una expresión como la siguiente:

```
pointcut getTotalPointcut(Purchase purchase) : execution(* Purchase.getTotal()) && target(purchase);
```

El parámetro "target" se refiere a una instancia de la clase a la cual se le aplica la operación.

En caso de que el método a interceptar posea argumentos, se deberá añadir el parámetro "args":

```
pointcut validatePurchaseDetailsPointcut(Purchase purchase, ValidationContext context) : execution(* Purchase.validateEnterPurchaseDetails (ValidationContext) ) && target(purchase) && args(context);
```

El parámetro "args" se refiere a instancias de clases que actúan como parámetros del método a interceptar.

Luego, se debe definir el advice a ejecutar para el pointcut definido.

Como ha sido mencionado, existen 3 tipos de advices:

- **Before:** El advice se ejecuta antes de los join points del pointcut y se define de la siguiente manera:

```
before(Purchase purchase): getTotalPointcut(purchase) {
purchase.methodB();
...
}
```

- **After:** El advice se ejecuta después de los join points del pointcut y se define de la siguiente manera:

```
after(Purchase purchase, ValidationContext context): validatePurchaseDetailsPointcut (purchase, context){
        purchase.methodC(context);
...
}
```

- **Around:** El advice se ejecuta alrededor de los join points del pointcut. Existen 2 alternativas cuando se utiliza un advice around:
    - Sobreescribir el código de los join points interceptados, lo cual se realiza de la siguiente manera:

```
BigDecimal around(Purchase purchase):getTotalPointcut(purchase) {
        BigDecimal total=0;
        for (int i=0; i < purchase.getItems().size() ;i++){
                Item item= purchase.getItems().get(i);
                total= total + (item.getPrice() * item.getQuantity()) ;
        }
        return total * 0,1;
}
```

    - Ejecutar dentro del cuerpo del advice el procesamiento original de los join points mediante la sintaxis especial: **proceed**() y añadir los cambios deseados antes y/o después de dicha ejecución. Esta opción es más escalable y se realiza de la siguiente manera:

```
BigDecimal around(Purchase purchase):getTotalPointcut(purchase) {
        return proceed() * 0,1;
}
```

Ejecución condicional de las intercepciones de los aspectos

Dado que en Java es un lenguaje estáticamente tipado, las modificaciones de los aspectos se introducen en tiempo de compilación. Este factor implica la necesidad de agregar sentencias condicionales para controlar su ejecución. De esta manera, se logra que a partir de la consulta del estado de determinada funcionalidad volátil (activado o desactivado), un método pueda comportarse como corresponde. Es decir, con las modificaciones pertinentes si la funcionalidad volátil se encuentra activada, y sin ellas en el caso contrario.

Los métodos inyectados no requieren de este tipo de sentencias condicionales, ya que no deberían ser invocados a menos que la funcionalidad volátil que los contiene esté activada. Por el contrario, los métodos interceptados sí lo precisan.

Para controlar la ejecución de la intercepción de un método debe agregarse un sentencia condicional "if" en el pointcut definido. Esto puede realizarse de la siguiente manera:

```
pointcut getTotalPointcut(Purchase purchase) : execution(* Purchase.getTotal()) && target(purchase) && if(org.springframework.webflow.samples.volatil.classes.VolatileFunctionalityServiceImpl.getInstance().isEnabled("discountVF")) ;
```

Nota: El parámetro “discountVF” es el nombre de la funcionalidad volátil de la que depende su ejecución este pointcut. Más adelante, en la sección 6.5.4.1, se describirá cómo definir las funcionalidades volátiles con sus correspondientes nombres.

6.5.3.2 Modificación de la capa de navegación

Los mecanismos utilizados para gestionar las modificaciones de la capa de navegación para este lenguaje de programación son semejantes a los utilizados para la capa de negocio (AOP). La diferencia fundamental entre las modificaciones de ambas capas radica en los componentes a los cuales comprenden.

Al igual que en la capa de negocio, será necesario ubicar las modificaciones a efectuar dentro del aspecto definido para la funcionalidad volátil correspondiente. En caso de aún no existir, deberá crearse un aspecto como ha sido explicado en la sección anterior.

Inyección de dependencias

En caso de querer inyectarle una dependencia (como un servicio) a una clase, puede realizarlo definiéndolo de la siguiente manera:

```
@Autowired
private  ExampleService ProductController.exampleService;
```

Inyección de métodos

Así como pueden inyectarse nuevas dependencias en una clase, también pueden inyectarse nuevos métodos.

A continuación se muestra como pueden inyectarse getters y setters para una nueva dependencia inyectada:

```
public BigDecimal ProductController.getExampleService() {
        return exampleService;
    }

public void ProductController.setExampleService(ExampleService es){
        this.exampleService = es;
}
```

Para definir un nuevo método en una interface o definir un método abstracto se debe definir un método con el cuerpo vacío como se muestra a continuación:

```
abstract  void ProductService.methodA() {}
```

Modificación del flujo original de métodos existentes

La modificación del flujo original de métodos existentes para la capa de navegación puede lograrse a partir de la intercepción de métodos. El mecanismo y la manera de realizarlo es idéntico al que ha sido explicado en la capa de negocio. Simplemente basta con definir pointcuts (con su sentencia condicional correspondiente) y advices asociados.

6.5.3.3 Modificación de la capa de presentación

Si lo que deseamos es modificar archivos de la capa de presentación, entonces utilizaremos transformaciones XSL que se ejecutarán en tiempo de compilación.

Cabe destacar que, por el momento, las vistas deberían ser escritas en XML y no en el tradicional JSP (basado en HTML), dado que estas últimas no son XMLs bien formados y no pueden ser parseadas por los transformadores de XML.

Con el objetivo de modificar las vistas, se debe definir un template de transformación por cada página XHTML, indicando a través de Saxon XSLT 2.0 en qué parte debe efectuarse dicha modificación. Cada uno de estos archivos XSL debe ser ubicado dentro de la carpeta src/volatile/resources/xsls.

Si la modificación de las vistas también involucra cambios en cuanto a estilos, entonces será necesario crear una nueva hoja de estilos que concentre todas las nuevas reglas. Esta hoja de estilos será nombrada "volatile-style.css", y deberá ser ubicada dentro de la carpeta src/volatile/resources/styles. Por supuesto, para que las nuevas reglas de estilos sean reflejadas, hará falta añadir (mediante una transformación xsl) la siguiente hoja de estilos al layout que corresponde:

```
<link rel="stylesheet" href="${request.contextPath}/styles/volatile-style.css" type="text/css"/>
```

En el Apéndice B, se describe cómo llevar a cabo la configuración y prueba del procesador XSLT.

Visualización condicional de las modificaciones

En este lenguaje de programación se implementa una visualización condicional de las modificaciones de esta capa. Por esta razón, se deben agregar sentencias condicionales en los archivos XSL para que las transformaciones de las vistas sean visualizadas sólo si determinada funcionalidad volátil se encuentra activada.

A continuación se muestra un ejemplo de :

```
<xsl:template match="//t:link[3]">
    //copia los tags originales contenidos
    <xsl:copy-of select="." />
<xsl:text disable-output-escaping="yes"><![CDATA[<c:if xmlns="http://java.sun.com/jsp/jstl/core" test="#{volatileFunctionalityService.map.volatileFunctionality.enabled}">

//tags a inyectar
<link xmlns="http://www.w3.org/1999/xhtml" rel="stylesheet"
```

```
href="${request.contextPath}/styles/volatile-style.css" type="text/css"/>

</c:if>]]></xsl:text>
</xsl:template>
```

En este caso, la hoja de estilos "volatile-style.css" sólo será incluida si la funcionalidad volátil "volatileFunctionality" se encuentra activada.

El código que ha sido resaltado es la condición que debería insertarse alrededor de cada una de las transformaciones definidas en el archivo XSL.

Se accede al mapa del servicio volatileFunctionalityService, luego a la funcionalidad deseada y por último al método enabled que devuelve el estado de la funcionalidad. El valor retornado finalmente definirá si la hoja de estilos debe ser incluida o no.

Dependiendo del tipo de transformación y de la manera en que la misma haya sido definida, puede que una sentencia condicional "if" resulte insuficiente. Si se modifica el contenido de un tag, por ejemplo agregándole un atributo nuevo, este tipo de sentencia no podría controlar su correcta inclusión. Consecuentemente, sería conveniente definir la transformación mediante una sentencia condicional "when.. otherwise". De esta manera, es posible definir el tag modificado a ser visualizado cuando la condición sea verdadera, y el tag original a ser visualizados cuando la condición sea falsa.

A continuación se muestra un ejemplo de uso de esta segunda alternativa:

```
<xsl:template match="node()[@id='banner']">
        <xsl:text disable-output-escaping="yes"><![CDATA[
        <c:choose xmlns="http://java.sun.com/jsp/jstl/core">
                <c:when test=
"#{volatileFunctionalityService.map.volatileFunctionality.enabled}">
        ]]></xsl:text>

                //tags modificados
 <h1 xmlns="http://www.w3.org/1999/xhtml" id="banner" class="primary">
        Funcionalidad volatil activada
</h1>

        <xsl:text disable-output-escaping="yes"><![CDATA[
                </c:when>
                <c:otherwise>
        ]]></xsl:text>
                        //copia los tags originales contenidos
                        <xsl:copy-of select="." />
        <xsl:text disable-output-escaping="yes"><![CDATA[
                </c:otherwise>
        </c:choose>
        ]]></xsl:text>
</xsl:template>
```

Por último, para que estas transformaciones funcionen correctamente deberíamos agregar, de la siguiente manera, el espacio de nombres xmlns:c="*http://java.sun.com/jsp/jstl/core*" tanto al archivo XSL en el encabezado, como al archivo resultante de la transformación. Esto se logra de la siguiente manera:

```
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:t="http://www.w3.org/1999/xhtml"
        xmlns:p="http://primefaces.org/ui"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:c="http://java.sun.com/jsp/jstl/core"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        exclude-result-prefixes="#all">
        ...

<xsl:template match="/*">
        <xsl:copy>
        <xsl:attribute                                    name="xmlns:c"
        namespace="http://java.sun.com/jsp/jstl/core"/>
                <xsl:apply-templates select="@* | node()"/>
                </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

6.5.4 Gestión de las modificaciones

6.5.4.1 Administración de las funcionalidades volátiles

La administración de las funcionalidades volátiles requiere la definición de determinadas clases y servicios.

Primero, se debe crear una clase mediante la cual se puedan representar las funcionalidades volátiles. Como se muestra a continuación, esta clase será llamada "VolatileFunctionality" y contendrá el atributo boolean "enabled" y los métodos necesarios para poder obtener y modificar el valor de este atributo.

```
public class VolatileFunctionality {

        private boolean enabled=true;

        public void enable(){
        enabled=true;
        }

        public void disable(){
        enabled=false;
        }

        public boolean getEnabled(){
```

```
        return enabled;
        }

        public void setEnabled(boolean enabled2){
        enabled=enabled2;
        }
}
```

Luego, se debe crear un servicio que implemente el patrón Singleton y permita la activación, desactivación y consulta del estado de cada una de las funcionalidades volátiles. Será el punto de entrada para acceder a estas funcionalidades.

Para ello crearemos la siguiente interface:

```
public interface VolatileFunctionalityService {

        public void enable(String key);

        public void disable(String key);

        public boolean isEnabled(String key);

        public HashMap<String,VolatileFunctionality> getMap();

        public void setMap(HashMap<String,VolatileFunctionality> map);

}
```

Posteriormente, crearemos el servicio que implemente esa interface. Este servicio poseerá un HashMap que contendrá cada funcionalidad volátil disponible, junto a un nombre que la identifique. En el siguiente caso se las han agregado 2 funcionalidades y se las ha nombrado “discountVF” y “exampleVF”.

```
@Service("volatileFunctionalityService")
public class VolatileFunctionalityServiceImpl implements
        VolatileFunctionalityService {

        private static VolatileFunctionalityServiceImpl INSTANCE = null;
        private HashMap<String, VolatileFunctionality> map = new
HashMap<String, VolatileFunctionality>();

        private VolatileFunctionalityServiceImpl() {
      map.put("discountVF", new VolatileFunctionality());
      map.put("exampleVF", new VolatileFunctionality());
        }

        static public VolatileFunctionalityService getInstance() {
```

```
                if (INSTANCE == null) {
                        INSTANCE = new VolatileFunctionalityServiceImpl();
                }
                return INSTANCE;
                }

                private VolatileFunctionality getFuseFor(String key) {
                return map.get(key);
                }

                synchronized public void enable(String key) {
                getFuseFor(key).enable();
                }

                synchronized public void disable(String key) {
                getFuseFor(key).disable();
                }

                synchronized public boolean isEnabled(String key) {
                return getFuseFor(key).getEnabled();
                }

        public HashMap<String, VolatileFunctionality> getMap() {
                return map;
                }

        public void setMap(HashMap<String,VolatileFunctionality> map) {
                this.map = map;
                }
}
```

La utilización de la palabra clave "synchronized" para el encabezado de los métodos públicos es fundamental dentro de este servicio. Al indicar que un método es sincronizado se está indicando que comparte una sección crítica. De este modo, cuando dos o más threads quieran acceder a él, no interferirán entre sí, sino que accederán de a uno a la vez.

6.5.4.2 Weaving de las funcionalidades volátiles

XSLT para archivos de configuración

Una aplicación web contiene archivos de configuración, y al introducir nuevas funcionalidades volátiles es probable que estos requieran cambios. Al igual que para la modificación de los archivos xhtmls de las vistas, las transformaciones XSL también pueden ser utilizados para los archivos de configuración XML.

Este es el caso del archivo de configuración de Spring "web-application-config.xml". Dado que las nuevas clases java, añadidas con la nueva funcionalidad volátil, pueden poseer anotaciones de Spring tales como @Service, debemos agregar el paquete que las contiene como component-scan para que los beans de Spring puedan ser creados.

Para ello crearemos primero un archivo XML que contendrá el tag “component-scan” necesario para que se encuentran las nuevas clases java. Llamaremos a este archivo “volatile-webapplication-config.xml” y lo ubicaremos en la carpeta src/volatile/resources/config. Este archivo será utilizado de aquí en más para agregar todas las configuraciones relacionadas a las funcionalidades volátiles. Por el momento el contenido de “volatile-webapplication-config.xml” será el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:component-scan base-package="org.springframework.webflow.samples.volatil.classes" />

</beans>
```

Lógicamente, el archivo de configuración “volatile-webapplication-config.xml” no será detectado automáticamente. Para que su contenido sea tenido en cuenta, deberemos crear un archivo XSL para modificar el contenido del archivo de configuración de la aplicación original (web-application-config.xml) y así importar el nuevo archivo de configuración creado (volatile-webapplication-config.xml).

Nombraremos a este nuevo archivo XSL “volatile_web-application-config.xsl” y lo ubicaremos en la carpeta src/volatile/resources/xsls como corresponde. Su contenido será el siguiente:

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:t="http://www.w3.org/1999/xhtml"
    xmlns:p="http://primefaces.org/ui"
xmlns:b="http://www.springframework.org/schema/beans"
    exclude-result-prefixes="b t p ">

    <xsl:output method="xml" encoding="utf-8" indent="yes" />

    <!-- identity rule -->
    <xsl:template match="node()|@*">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*" />
        </xsl:copy>
    </xsl:template>

    <xsl:template match="//b:import[last()]">
```

```
                <xsl:copy-of select="." />
                <import resource="volatile-webapplication-config.xml"
                xmlns="http://www.springframework.org/schema/beans" />
        </xsl:template>

</xsl:stylesheet>
```

Maven

Maven es el responsable de lanzar la compilación de nuestro sistema y de crear el archivo de distribución War.

Para poder adicionar las funcionalidades volátiles a nuestro sistema de una manera no intrusiva, se decidió utilizar un contexto diferente de compilación, modificando el ciclo de compilación clásico de Maven. Dado el proyecto original, se lo abre, se le agregan los aspectos, se realizan las transformaciones XSL y se lo vuelve a empaquetar.

Para que esto sea posible, el archivo pom.xml deberá definir un build como el siguiente, en el cual se definan todos los plugins necesarios para modificar el ciclo de compilación de la aplicación:

```
<build>
<finalName>${project.artifactId}</finalName>
<plugins>
                <!—Plugins-->
        <plugins>
</build>
```

El valor "${project.artifactId}" de finalName hace referencia al artifactId del proyecto. Recordemos que el artifactId del proyecto "vf" debe ser el mismo que el de la aplicación original, para así poder mantener la url de la aplicación original.

A continuación se presentarán los plugins necesarios:

- **maven-dependency-plugin:** Este plugin es utilizado para manipular artifacts. En este caso se lo utiliza para poder desempaquetar el proyecto original definido como dependencia en el proyecto "vf", para así poder modificarlo. A continuación se muestra su configuración:

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <version>2.8</version>
        <executions>
                <execution>
                <id>unpack</id>
                <phase>generate-sources</phase>
                <goals>
```

```
                    <goal>unpack</goal>
            </goals>
            <configuration>
            <artifactItems>
            <artifactItem>
            <groupId>org.springframework.webflow.samples</groupId>
                    <artifactId>app-web</artifactId>
            <type>war</type>
            <version>1.0.0.BUILD-SNAPSHOT</version>
<outputDirectory>${project.build.directory}/${project.artifactId}
</outputDirectory>
            </artifactItem>
            </artifactItems>
            <!-- other configurations here -->
            </configuration>
            </execution>
            </executions>
</plugin>
```

- **build-helper-maven-plugin:** Este plugin es utilizado para setear los directorios que contienen los aspectos y las nuevas clases java, como source folders en tiempo de compilación. A continuación se muestra su configuración:

```
<plugin>
<groupId>org.codehaus.mojo</groupId>
        <artifactId>build-helper-maven-plugin</artifactId>
        <version>1.8</version>
        <executions>
                <execution>
                        <id>add-source</id>
                        <phase>generate-sources</phase>
                        <goals>
                                <goal>add-source</goal>
                        </goals>
                        <configuration>
                                <sources>
                           <source>src/volatile/java</source>
                           <source>src/volatile/aspects</source>
                        </sources>
                        </configuration>
                </execution>
</executions>
</plugin>
```

- **xml-maven-plugin:** Tiene la función de realizar la transformación XSL a partir de los templates explicados anteriormente. También se agrega la

dependencia para el transformador XSL Saxon. A continuación se muestra su configuración:

```
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>xml-maven-plugin</artifactId>
        <executions>
                <execution>
                        <phase>generate-sources</phase>
                        <goals>
                                <goal>transform</goal>
                        </goals>
                </execution>
        </executions>
<configuration>
                <transformationSets>
    <!—Transformaciones xsl correspondientes al perfil volatile -->
                ...
        </transformationSets>
</configuration>
<dependencies>
        <dependency>
                        <groupId>net.sf.saxon</groupId>
                <artifactId>saxon</artifactId>
                        <version>8.7</version>
        </dependency>
</dependencies>
</plugin>
```

Cada una de las transformaciones XSL se define de la siguiente manera:

```
<transformationSet>
<dir>${project.build.directory}/${project.artifactId}/WEB-INF/layouts
</dir>
        <outputDir>
        ${project.build.directory}/xsls/WEB-INF/layouts
</outputDir>
        <includes>
                <include>home.xhtml</include>
        </includes>
<stylesheet>
        src/volatile/resources/xsls/volatile_home.xsl
</stylesheet>
</transformationSet>
```

- **dir:** contiene la ubicación de la carpeta en la que se encuentra el archivo que se desea transformar.

- **outputDir:** contiene la carpeta en la que se ubicará el nuevo archivo transformado. El mismo debe ubicarse dentro de ${project.build.directory}/xsls/WEB-INF/carpeta en la que se encuentra el archivo original dentro de WEB-INF(dir).
- **include:** contiene el archivo original que será transformado.
- **stylesheet:** contiene el archivo XSL que será utilizado para transformar el archivo original. Este archivo se encuentra dentro del proyecto "vf".

Nota: No debemos olvidarnos de agregar la transformación XSL necesaria para añadir el tag "component-scan" en el archivo web-application-config.xml de Spring. Esta transformación puede definirse de la siguiente manera:

```
<transformationSet>
	<dir>${project.build.directory}/${project.artifactId}/WEB-INF/config
</dir>
	<outputDir>
	${project.build.directory}/xsls/WEB-INF/config
	</outputDir>
	<includes>
		<include>web-application-config.xml</include>
</includes>
	<stylesheet>
src/volatile/resources/xsls/volatile_web-application-config.xsl
	</stylesheet>
</transformationSet>
```

- **maven-merge-properties-plugin:** Se hace uso del plugin "Maven Merge Properties" para adicionar a los archivos de i18n del sistema los mensajes que corresponden a la funcionalidad volátil. El plugin funciona indicando cuáles son los archivos de mensajes requeridos (del proyecto original y de la funcionalidad volátil)y devuelve un nuevo archivo a partir de un merge de los mismos.
  De esta forma, el archivo original no debe ser modificado para adicionar los nuevos mensajes. A continuación se muestra su configuración:

```
<plugin>
<groupId>org.beardedgeeks</groupId>
<artifactId>maven-merge-properties-plugin</artifactId>
<version>0.2</version>
<configuration>
<merges>
	<merge>
		<targetFile>
${project.build.directory}/messages/WEB-
```

```
INF/messages/messages.properties
                    </targetFile>
                    <propertiesFiles>
                        <propertiesFile>
    ${project.build.directory}/${project.artifactId}/WEB-
    INF/messages.properties
                        </propertiesFile>
                <propertiesFile>
                src/volatile/resources/properties/volatileReq.properties
                        </propertiesFile>
                        </propertiesFiles>
                </merge>
        </merges>
        </configuration>
        <executions>
                <execution>
                        <phase>prepare-package</phase>
                        <goals>
                                <goal>merge</goal>
                        </goals>
                </execution>
        </executions>
        </plugin>
```

- **targetFile:** contiene la ubicación que tendrá el archivo generado por la combinación.
- **propertiesFiles:** Contiene los archivos de mensajes a combinar (originales y de la funcionalidad volátil).

- **maven-war-plugin:** Este plugin es el responsable de recoger dependencias, clases y recursos de la aplicación web y empaquetarlos dentro del archivo web de la aplicación. Es necesario utilizarlo para incluir los XSLs y mensajes (que reemplazarán a los originales), y demás recursos como imágenes, hojas de estilos y xhtmls (en los cuales se indicará el path de destino que tendrán). A continuación se muestra su configuración:

```
        <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
        <webResources>
                <resource>
                        <directory>
                        ${project.build.directory}/xsls
                </directory>
                </resource>
```

```
            <resource>
                <directory>
                 src/volatile/resources/config
            </directory>
                <targetPath>WEB-INF/config</targetPath>
            </resource>

            <resource>
            <directory>
                ${project.build.directory}/messages
            </directory>
            </resource>
            <resource>
            <directory>
                src/volatile/resources/xhtml
            </directory>
                <targetPath>WEB-INF</targetPath>
            </resource>
            <resource>
                <directory>
                src/volatile/resources/images
            </directory>
                <targetPath>images</targetPath>
            </resource>
            <resource>
                <directory>
                src/volatile/resources/styles
            </directory>
                <targetPath>styles</targetPath>
            </resource>
            <resource>
                <directory>
                src/volatile/resources/rules
            </directory>
                <targetPath>WEB-INF/classes/rules</targetPath>
            </resource>
        </webResources>
        </configuration>
        </plugin>
```

- **aspectj-maven-plugin:** Modifica las clases compilada a partir de los aspectos explicados anteriormente. A continuación se muestra su configuración:

```
        <plugin>
        <groupId>org.codehaus.mojo</groupId>
            <artifactId>aspectj-maven-plugin</artifactId>
```

```
            <configuration>
                <source>1.6</source>
                <complianceLevel>1.6</complianceLevel>
            <weaveDependencies>
                    <weaveDependency>
                    <groupId>
                     org.springframework.webflow.samples
                </groupId>
                    <artifactId>app-web</artifactId>
                    <classifier>classes</classifier>
                    </weaveDependency>
                </weaveDependencies>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                     <goal>test-compile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
```

- **maven-jetty-plugin:** Configura el servidor web Jetty para que la aplicación sea ejecutada dentro en él. A continuación se muestra su configuración:

```
        <plugin>
            <groupId>org.mortbay.jetty</groupId>
            <artifactId>maven-jetty-plugin</artifactId>
            <configuration>
                <scanIntervalSeconds>10</scanIntervalSeconds>
                <connectors>
                <connector
implementation="org.mortbay.jetty.nio.SelectChannelConnector">
            <port>8080</port>
            <maxIdleTime>60000</maxIdleTime>
            </connector>
            </connectors>
            </configuration>
        </plugin>
```

6.5.4.3 Subetapas de las funcionalidades volátiles

Es posible definir subetapas para las funcionalidades volátiles. Para ello, la clase "VolatileFunctionality" debería redefinir los métodos #enable() y #disable(). Además, tanto la clase "VolatileFunctionality" como los métodos #enableSetUp() y #preDisable() deberían ser definidos como abstractos. De esta manera, las funcionalidades volátiles a implementar deberían extender de "VolatileFunctionality" y definir estos métodos. A continuación se muestran todos estos cambios introducidos en la clase "VolatileFunctionality".

```
abstract class VolatileFunctionality {

        private boolean enabled=true;

        public void enable(){
                enabled=true;
                this.enableSetUp();
        }

        public void disable(){
                this.preDisable();
                enabled=false;
        }

        public boolean getEnabled(){
                return enabled;
        }

        public void setEnabled(boolean enabled2){
                enabled=enabled2;
        }

        abstract void enableSetUp();

        abstract void preDisable();
}
```

6.5.4.4 Commons para funcionalidades volátiles

El concepto de "Commons" para funcionalidades volátiles no ha sido cubierto para este lenguaje de programación. El mismo no ha sido precisado por el prototipo y no ha podido ser probado como corresponde. No obstante, no debería haber ningún inconveniente para llevar a cabo su implementación. Para lograr este objetivo debería ser suficiente con:

- Modificar la estructura de las funcionalidades volátiles para poder incluir funcionalidades como "Commons".

- Definir un nuevo método a extender para incluir las funcionalidades agregadas como “Commons”.
- Modificar el comportamiento de los métodos #enable() y #disable() teniendo en cuenta las consideraciones necesarias.

6.5.4.5 Programación de eventos de activación y desactivación

Hasta el momento, a lo largo de la guía se han descrito todos los pasos a seguir para lograr el desacoplamiento de las funcionalidades volátiles en una aplicación web Java. Mediante la utilización de un motor de reglas, además se brinda la posibilidad de programar la activación y la desactivación de estas funcionalidades volátiles.

Drools [9] es un sistema de administración de reglas de negocio (BRMS) con un motor de reglas basado en una adaptación orientada a objetos del algoritmo Rete. Permite expresar de forma más natural las reglas de negocio interactuando con los objetos de negocio.

Drools permite definir reglas en archivos de reglas DRL. Cada uno de estas reglas está compuesta por un conjunto de condiciones y/o expresiones y por acciones a ejecutar. Si la evaluación de la expresión de una regla es true, entonces se ejecutarán las acciones programadas y efectivamente una funcionalidad volátil será activada o desactivada.

En los archivos de reglas, se deberán definir todos aquellos momentos y eventos ante los que se desea que las distintas funcionalidades volátiles sean activadas y/o desactivadas. Es posible configurar para las reglas tanto fechas y horarios, como aún también la evaluación de ciertas condiciones predefinidas.

Esta posibilidad resulta útil e interesante, ya que al permitir configurar con antelación la activación y desactivación de las funcionalidades volátiles, los cambios correspondientes serán reflejados en tiempo de ejecución, sin necesidad de volver a compilar y correr la aplicación.

Para incorporar esta capacidad deberemos:

- **Modificar el archivo de configuración XML:** Modificar el archivo de configuración “volatile-webapplication-config.xml” (situado en la carpeta src/volatile/resources/config) para agregar la configuración necesaria para el motor de reglas Drools y el bean para el servicio mediante el cual se manejarán las funcionalidades volátiles. A continuación se muestra la configuración resultante:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:drools="http://drools.org/schema/drools-spring"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://drools.org/schema/drools-spring http://drools.org/schema/drools-
spring.xsd">
```

```
        <context:component-scan base-
package="org.springframework.webflow.samples.volatil.classes" />

            <drools:grid-node id="node1" />
            <drools:kstore id="kstore1">

            </drools:kstore>
            <drools:resource id="resource1" type="DRL"
        source="classpath:rules/volatileFunctionalitiesRules.drl" />
            <drools:kbase id="kbase1" node="node1">

            <drools:resources>
                    <drools:resource ref="resource1" />
            </drools:resources>
            </drools:kbase>
            <drools:ksession id="ksession1" type="stateful" kbase="kbase1"
        node="node1">

            <drools:agendaEventListener>
            <bean class= "org.drools.event.rule.DebugAgendaEventListener">
            </bean>

            </drools:agendaEventListener>
            <drools:workingMemoryEventListener>
            <bean class=
            "org.drools.event.rule.DebugWorkingMemoryEventListener">
            </bean>

            </drools:workingMemoryEventListener>
            <drools:batch>
                    <drools:fire-all-rules />
                    <drools:set-global identifier="volatileFunctionalityService"
                ref="volatileFunctionalityService" />
            </drools:batch>
            </drools:ksession>

            <bean id="volatileFunctionalityService"
        class="org.springframework.webflow.samples.volatil.classes.VolatileFuncti
        onalityServiceImpl" factory-method="getInstance">
            </bean>

        </beans>
```

- **Crear reglas:** Crear un archivo DRL en el cual se definan las reglas que lanzarán las activaciones y desactivaciones de las funcionalidades volátiles. Nombraremos a este archivo “volatileFunctionalitiesRules.drl” y lo ubicaremos en la carpeta src/volatile/resources/rules. Si este directorio no existe aún, deberemos crearlo.

Una vez creado el archivo de reglas, lo que haremos será definir la variable global "volatileFunctionalityService" como se muestra a continuación. La precisaremos para poder acceder al servicio "VolatileFunctionalityService" y a sus métodos dentro de las reglas que definamos.

```
global VolatileFunctionalityService volatileFunctionalityService;
```

Para definir una regla se utilizará la siguiente plantilla:

```
rule "nombre de la regla"
date-effective "12-Feb-2015"
date-expires "14-Feb-2015"
when
        //Condicion
eval(true);
then
        //Acciones a ejecutar
end
```

- **rule:** Nombre de la regla.
- **date-effective:** Fecha de inicio desde la que la regla será evaluada.
- **date-expires:** Fecha de fin hasta la que la regla será evaluada.

Para programar que una funcionalidad volátil esté disponible por un tiempo determinado, se deben definir dos reglas:

- **Regla de activación:**

```
rule "discountVF activation rule"
date-effective "12-Feb-2015"
date-expires "14-Feb-2015"
when
        eval(true);
then
        System.out.println("discountVF enable");
        volatileFunctionalityService.enable("discountVF");
 end
```

- **Regla de desactivación:**

```
rule "discountVF deactivation rule"
date-effective "14-Feb-2015"
date-expires "16-Feb-2015"
when
        eval(true);
then
System.out.println("discountVF disable");
        volatileFunctionalityService.disable("discountVF");
end
```

La regla de desactivación debe programarse para ser evaluada una vez que la regla de activación ha expirado. Esto se consigue haciendo coincidir el date-effective de la regla de activación con el date-expires de la regla de desactivación.

6.5.4.6 Ejecución

Como fue explicado, podemos optar entre ejecutar la aplicación web con o sin funcionalidades volátiles.

A continuación se detallan los pasos a seguir para ambas ejecuciones:

6.5.4.6.1 Ejecución de la aplicación original

1) Acceder a la perspectiva de Spring (Menú Windows > Open Perspective >Other, Spring).
2) Seleccionar el proyecto original, hacer click derecho sobre el proyecto original, seleccionar “configure” y luego “Convert to maven Project”.
3) Seleccionar el proyecto original, hacer click derecho y seleccionar Run as > maven install.
4) Seleccionar el proyecto original, hacer click derecho y luego seleccionar Run as > maven build.. , ingresar “tomcat:run” en el campo goals, y por último presionar el botón “Run”.
5) Abrir la URL http://localhost:8080/appWeb con un navegador.

Nota: http://localhost:8080/**appWeb** debería ser reemplazado por el nombre de la aplicación que se desea correr.

6.5.4.6.2 Ejecución de la aplicación con las funcionalidades volátiles

1) Acceder a la perspectiva de Spring (Menú Windows > Open Perspective >Other, Spring).
2) Seleccionar el proyecto original, hacer click derecho sobre el proyecto original, seleccionar “configure” y luego “Convert to maven Project”.
3) Seleccionar el proyecto original, hacer click derecho y seleccionar Run as > maven install.

4) Seleccionar el proyecto vf, hacer click derecho sobre el proyecto original, seleccionar “configure” y luego “Convert to maven Project”.
5) Seleccionar el proyecto vf, hacer click derecho y seleccionar Run as > maven install.
6) Seleccionar el proyecto vf, hacer click derecho y luego seleccionar Run as > maven build.. , ingresar “clean jetty:run-war” en el campo goals, y por último presionar el botón “Run”.
7) Abrir la URL http://localhost:8080/appWeb con un navegador.

Nota: http://localhost:8080/**appWeb** debería ser reemplazado por el nombre de la aplicación que se desea correr.

7 Implementación en Smalltalk

7.1 Objetivo

Se ha desarrollado el VF Framework para Smalltalk con el propósito de probar y proveer un Framework para la implementación y manipulación de funcionalidades volátiles para este lenguaje de programación. El mismo tiene como objetivo la simplificación, automatización y mejora de los procesos para llevarlo a cabo.

7.2 Tecnologías utilizadas

Para la implementación de funcionalidades volátiles siguiendo el VF Framework para el lenguaje de programación Smalltalk, se han utilizado las siguientes tecnologías:

- Pharo Smalltalk 2.0 [32]
- Seaside 3.1[17]
- PHANtom (PHaro Aspect laNguage) 1.2.3 [18]
- Scheduler SeanDeNegris.22 [19]

7.3 Implementación

Aprovechando las ventajas que provee un lenguaje de programación orientado a objetos, dinámicamente tipado y potente como lo es Smalltalk, se desarrolló un Framework de caja blanca llamado "VF Framework". Este Framework para Smalltalk contiene todo el código necesario para la manipulación de funcionalidades volátiles en aplicaciones web con Seaside. Simplemente a partir de la extensión de determinadas clases y de la implementación de determinados hotspots, resulta posible gestionar la incorporación desacoplada y la activación y desactivación programada de funcionalidades volátiles.

Para cumplir con el objetivo deseado de incorporar funcionalidades volátiles, como VF Framework propone, se decidió:

- Mantener sin alterar la aplicación original
- Proveer un Framework de caja blanca para la manipulación de las funcionalidades volátiles con hotspots a extender
- Implementar un nuevo proyecto con las funcionalidades volátiles que decoran a la aplicación original
- Utilizar AOP (en lenguaje dinámicamente tipado)
- Utilizar weaving dinámico
- Utilizar un Scheduler
- Utilizar modificaciones mediante expresiones regulares

Mediante este tipo de implementación resulta una tarea muy sencilla el proceso de correr la aplicación original con o sin las funcionalidades volátiles añadidas. Además, el Framework provisto también facilita el proceso de implementación y configuración de funcionalidades volátiles.

Las funcionalidades volátiles son añadidas a partir de aspectos. Phantom es un lenguaje de aspectos dinámicos para Smalltalk que posibilita la incorporación de variables y métodos, así también como la intercepción de métodos a partir de conjuntos

de pointcuts y advices asociados. Estos aspectos pueden ser instalados y desinstalados en tiempo de ejecución, según sea requerido.

La intercepción de métodos es un factor muy importante a la hora de implementar funcionalidades volátiles. Sin embargo, cuando deben coexistir múltiples funcionalidades volátiles que involucran modificaciones de secciones de código coincidentes, la intercepción puede conllevar ciertas limitaciones. Los advices pueden ser definidos para ejecutarse antes, después o en el momento de acceder a determinado pointcut. Esto puede resultar insuficiente y está limitado por el grado de modularización que un método posea. Por esta razón, se ha provisto una solución adicional como lo es la modificación de métodos a través de expresiones regulares.

Otra característica fundamental que posee el enfoque presentado es la posibilidad de programar la activación y la desactivación de las funcionalidades volátiles. Para ello se utiliza AOP (en un lenguaje dinámicamente tipado) y un Scheduler. Los eventos temporales utilizan el Scheduler, los eventos de negocio utilizan AOP dinámico y los eventos de negocio temporales utilizan una combinación del Scheduler y AOP dinámico. Sin embargo, la manipulación interna de los eventos es transparente al programador, quien simplemente configura los eventos a partir de una serie de métodos predefinidos por el Framework.

7.4 Prototipo desarrollado

7.4.1 Descripción

Con el objetivo de probar el marco de trabajo presentado, se implementó un prototipo para el lenguaje de programación Smalltalk haciendo uso del Framework para aplicaciones web Seaside.

Este prototipo ha sido desarrollado extendiendo el VF Framework. Por un lado se mantuvo el proyecto original, mientras que en otro proyecto aparte se ubicó todo el código perteneciente a las funcionalidades volátiles a incorporar.

Se tomó como base para el desarrollo del prototipo a la aplicación web SushiNet Store. SushiNet Store es una aplicación simple de una tienda de delivery de sushi, provista como sample de Seaside. Este sample cuenta con un carrito de compras y simula la compra de productos; sin embargo no cuenta con una base de datos real. En el proceso de compra, primero se deben seleccionar y agregar al carrito los productos deseados (a partir del listado o la búsqueda de los mismos), y luego se deben completar los datos correspondientes a la dirección de envío, a la dirección de pago y a la tarjeta de crédito del cliente solicitante.

El sample SushiNet Store fue obtenido de la imagen "Seaside One-Click Experience 2.8.4", disponible en [32]. Esta imagen posee el IDE Pharo Smalltalk versión 0.1 y Seaside versión 2.8.4. Luego, se descargó una imagen más actualizada con Pharo Smalltalk 2.0 y Seaside 3.1. Debido a la aparición de ciertas incompatibilidades al ser portado a una imagen más actualizada, el sample tuvo que ser adaptado para lograr un correcto funcionamiento. No obstante, más allá de los cambios mínimos que han sido realizados para la portación de la aplicación, el único cambio adicional añadido ha sido un contador para registrar la cantidad de ventas realizadas.

Para este prototipo se han incorporado las siguientes funcionalidades volátiles:

- **Promoción de Atún (TunaPromotionVF):** La tienda SushiNet ha decidido lazar una promoción para promover las ventas de todos los productos que contienen atún entre sus ingredientes. Cuando dicha promoción se encuentre activa, todos los productos que poseen atún

tendrán un descuento del 20%. Esta promoción alternará entre activada y desactivada cada 5 minutos.
Dentro del conjunto de cambios introducidos por este requisito podemos encontrar:

- Banner de la promoción en la cabecera de las páginas. El mismo se encuentra resaltado con un rectángulo color rojo en la Ilustración 43.

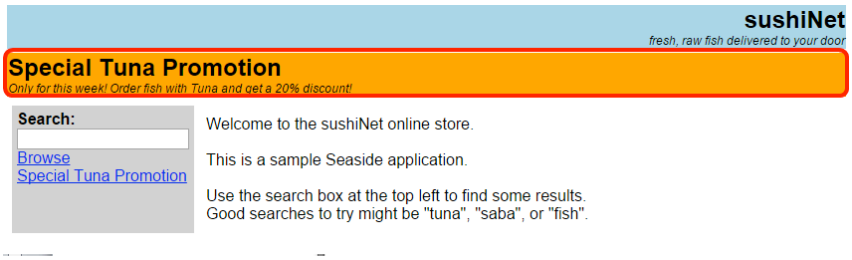


Ilustración 43: Prototipo Smalltalk – TunaPromotionVF – Banner

- Link debajo de la caja de búsqueda para obtener el listado de los productos promocionados, el cual se encuentra resaltado con un rectángulo color rojo en la Ilustración 44.

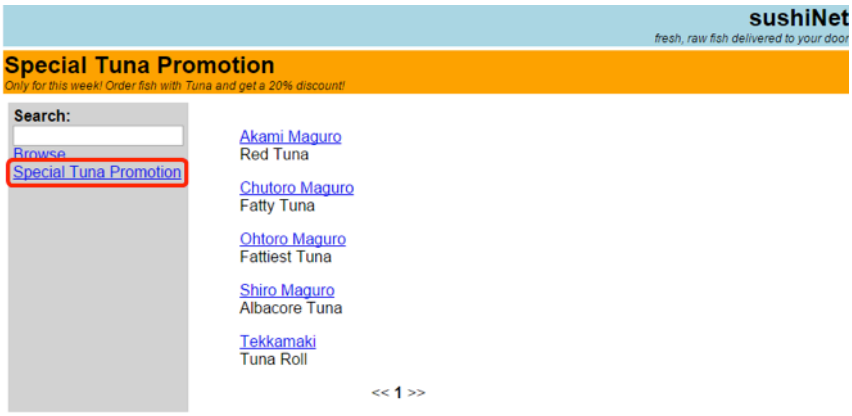


Ilustración 44: Prototipo Smalltalk – TunaPromotionVF – Link

- Detalle de los productos promocionados enriquecido. El detalle de los productos promocionados muestra el precio sin promoción tachado en color rojo, el nuevo precio con la promoción y el descuento existente. Esto puede observarse resaltado con un rectángulo color rojo en la Ilustración 45.

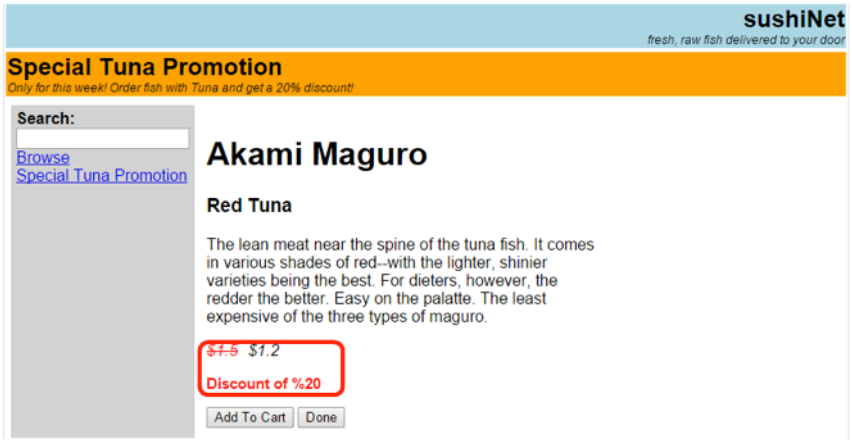


Ilustración 45: Prototipo Smalltalk – TunaPromotionVF – Detalle enriquecido

- El monto total visualizado contempla la aplicación de la promoción. Esto puede observarse resaltado con rectángulos color rojo en la Ilustración 46 y en la Ilustración 47.

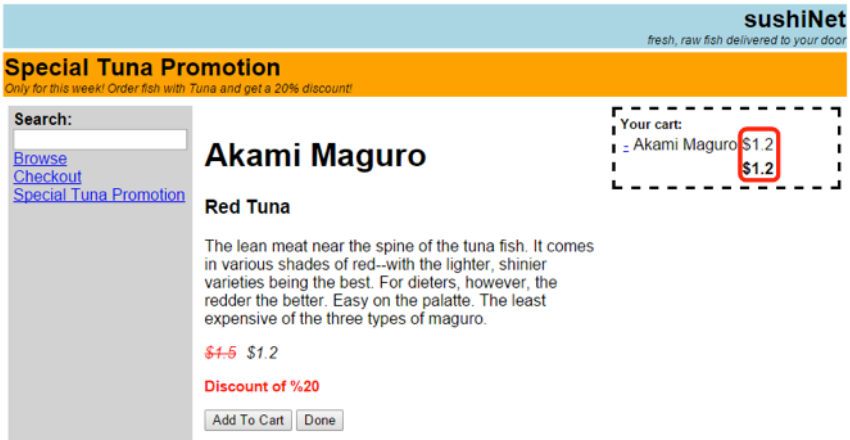


Ilustración 46: Prototipo Smalltalk – TunaPromotionVF – Monto total en carrito

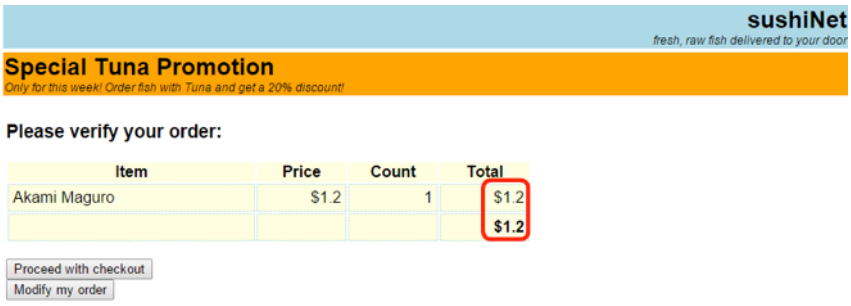


Ilustración 47: Prototipo Smalltalk – TunaPromotionVF – Monto total en columna

- **Promoción de Salmón (SalmonPromotionVF):** La tienda SushiNet ha decidido lazar una promoción para promover las ventas de todos los productos que contienen salmón entre sus ingredientes. Cuando dicha promoción se encuentre activa, todos los productos que poseen salmón tendrán un descuento del 10%. Esta promoción se activará por única vez 10 minutos luego de que se inicie la ejecución y se desactivará cuando se

hayan alcanzado las 10 ventas de estos productos con descuentos. Dentro del conjunto de cambios introducidos por este requisito podemos encontrar:

- Banner de la promoción en la cabecera de las páginas. El mismo se encuentra resaltado con un rectángulo color rojo en la Ilustración 48.

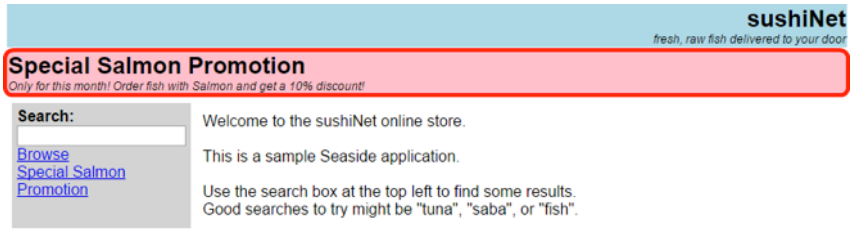


Ilustración 48: Prototipo Smalltalk – SalmonPromotionVF – Banner

- Link debajo de la caja de búsqueda para obtener el listado de los productos promocionados, el cual se encuentra resaltado con un rectángulo color rojo en la Ilustración 49.

Ilustración 49: Prototipo Smalltalk – SalmonPromotionVF – Link

- Detalle de los productos promocionados enriquecido. El detalle de los productos promocionados muestra el precio sin promoción tachado en color rojo, el nuevo precio con la promoción y el descuento existente. Esto puede observarse resaltado con un rectángulo color rojo en la Ilustración 50.

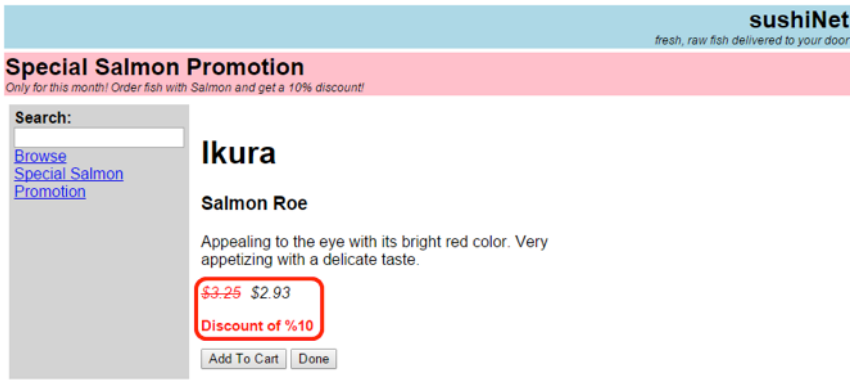


Ilustración 50: Prototipo Smalltalk – SalmonPromotionVF – Detalle enriquecido

- El monto total visualizado contempla la aplicación de la promoción. Esto puede observarse resaltado con rectángulos color rojo en la Ilustración 51 y en la Ilustración 52.

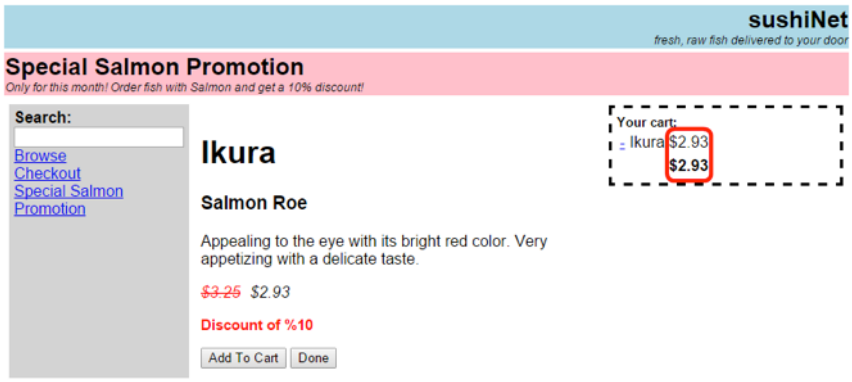


Ilustración 51: Prototipo Smalltalk – SalmonPromotionVF – Monto total en carrito

Ilustración 52: Prototipo Smalltalk – SalmonPromotionVF – Monto total en columna

- **Promoción de Roe (RoePromotionVF):** La tienda SushiNet ha decidido lazar una promoción para promover las ventas de todos los productos de tipo Roe. Cuando dicha promoción se encuentre activa, todos los productos de tipo Roe serán promocionados con una oferta de "Lleve 3 Pague 2". Esta promoción se activará cuando se llegue a las 2 ventas concretadas (sin importar los productos seleccionados), y se desactivará 2 minutos luego de ser activada.
  Dentro del conjunto de cambios introducidos por este requisito podemos encontrar:

  - Banner de la promoción en la cabecera de las páginas. El mismo se encuentra resaltado con un rectángulo color rojo en la Ilustración 53.

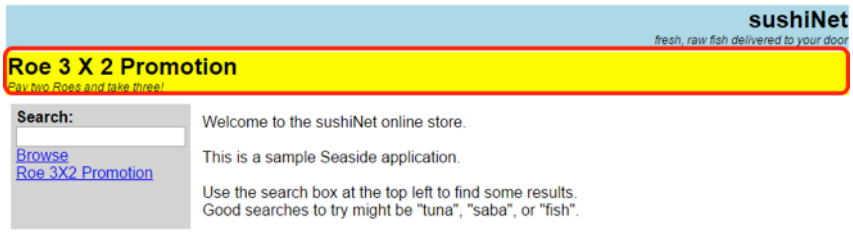


Ilustración 53: Prototipo Smalltalk – RoePromotionVF – Banner

- Link debajo de la caja de búsqueda para obtener el listado de los productos promocionados, el cual se encuentra resaltado con un rectángulo color rojo en la Ilustración 54.

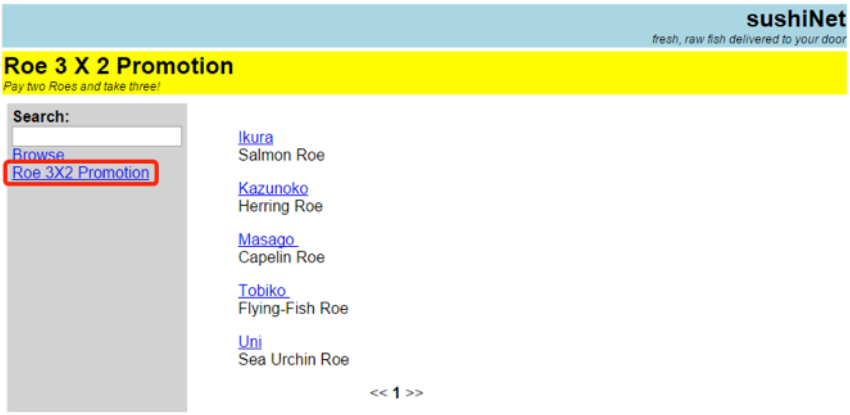


Ilustración 54: Prototipo Smalltalk – RoePromotionVF – Link

- Detalle de los productos promocionados enriquecido. El detalle de los productos promocionados muestra la promoción existente. Esto puede observarse resaltado con un rectángulo color rojo en la Ilustración 55.

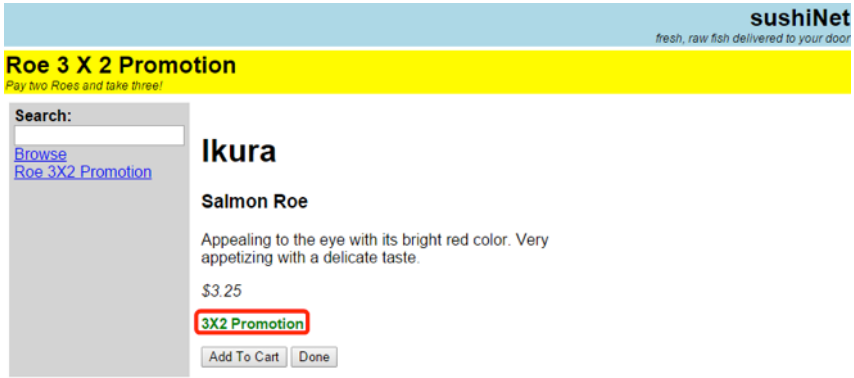


Ilustración 55: Prototipo Smalltalk – RoePromotionVF – Detalle enriquecido

- Listado de los productos del carrito enriquecido. En el listado de los productos del carrito se visualiza junto a los productos que entran en esta promoción, la cantidad de promociones aplicadas. Esto puede observarse resaltado con un rectángulo color rojo en la Ilustración 56.

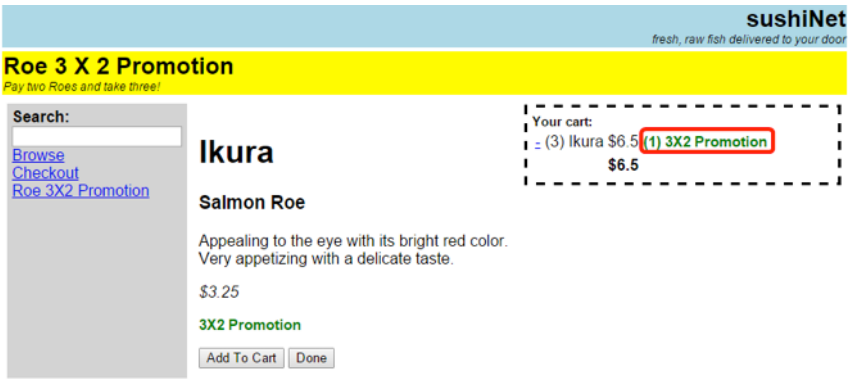


Ilustración 56: Prototipo Smalltalk – RoePromotionVF – Promociones en carrito

- El monto total visualizado contempla la aplicación de la promoción. Esto puede observarse resaltado con rectángulos color rojo en la Ilustración 57 y en la Ilustración 58.

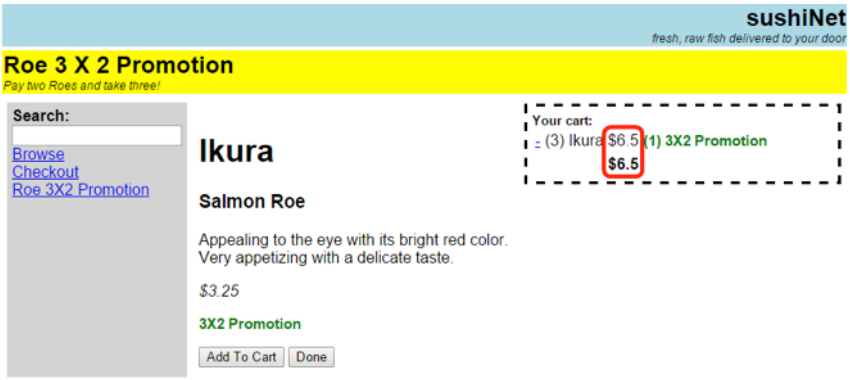


Ilustración 57: Prototipo Smalltalk – RoePromotionVF – Monto total en carrito

Ilustración 58: Prototipo Smalltalk – RoePromotionVF – Monto total en columna

- **Promoción de Roll (RollPromotionVF):** La tienda SushiNet ha decidido lazar una promoción para promover las ventas de todos los productos de tipo Roll. Cuando dicha promoción se encuentre activa, todos los productos de tipo Roll serán promocionados con una oferta de "Lleve 3 x Pague 2". Esta promoción se activará cuando la promoción de Roe sea desactivada y se desactivará cuando la promoción de Roll sea activada.

Dentro del conjunto de cambios introducidos por este requisito podemos encontrar:

- Banner de la promoción en la cabecera de las páginas. El mismo se encuentra resaltado con un rectángulo color rojo en la Ilustración 59.

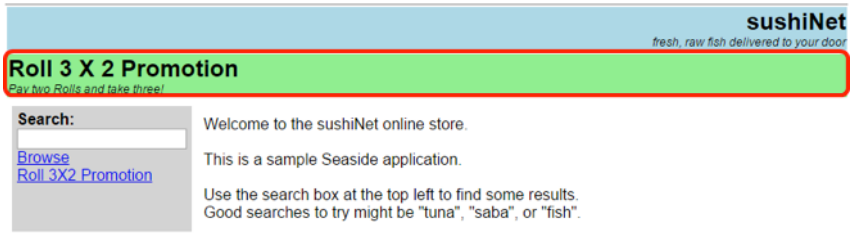


Ilustración 59: Prototipo Smalltalk – RollPromotionVF – Banner

- Link debajo de la caja de búsqueda para obtener el listado de los productos promocionados, el cual se encuentra resaltado con un rectángulo color rojo en la Ilustración 60.

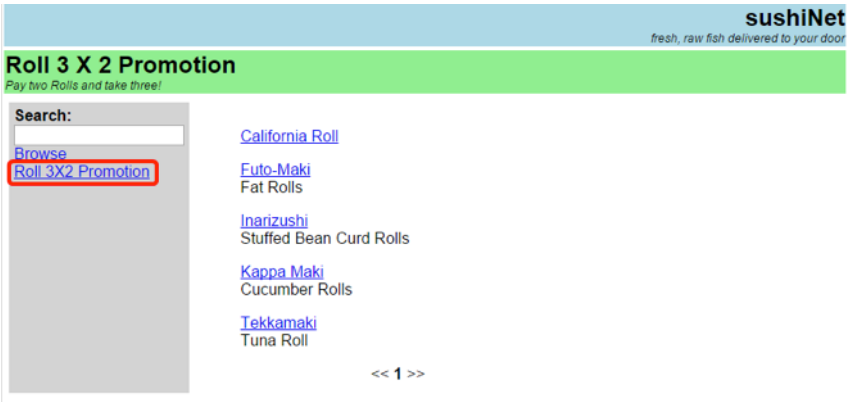


Ilustración 60: Prototipo Smalltalk – RollPromotionVF – Link

- Detalle de los productos promocionados enriquecido. El detalle de los productos promocionados muestra la promoción existente. Esto puede observarse resaltado con un rectángulo color rojo en la Ilustración 61.

Ilustración 61: Prototipo Smalltalk – RollPromotionVF – Detalle enriquecido

- Listado de los productos del carrito enriquecido. En el listado de los productos del carrito se visualiza junto a los productos que entran en esta promoción, la cantidad de promociones aplicadas. Esto puede observarse resaltado con un rectángulo color rojo en la Ilustración 62.

Ilustración 62: Prototipo Smalltalk – RollPromotionVF – Promociones en carrito

- El monto total visualizado contempla la aplicación de la promoción. Esto puede observarse resaltado con rectángulos color rojo en la Ilustración 63 y en la Ilustración 64.

Ilustración 63: Prototipo Smalltalk – RollPromotionVF – Monto total en carrito

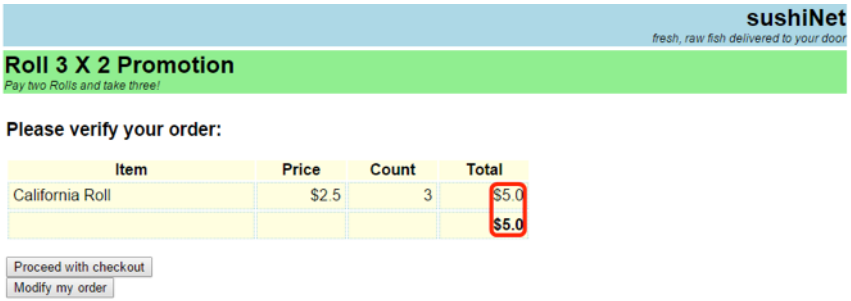


Ilustración 64: Prototipo Smalltalk – RollPromotionVF – Monto total en columna

Nota: Como puede observarse en la Ilustración 65, múltiples funcionalidades volátiles pueden estar activadas simultáneamente.

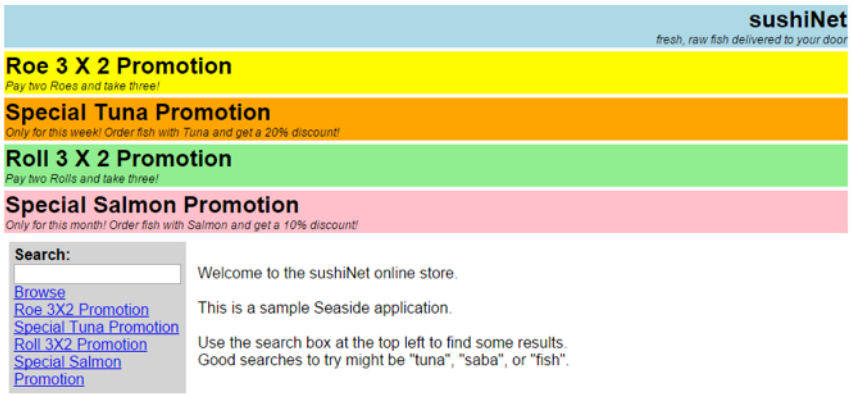


Ilustración 65: Prototipo Smalltalk – Todas las FVs activadas

7.4.2 Ejecución

En el Apéndice A, se describe cómo llevar a cabo la instalación del prototipo desarrollado.

Como será explicado a lo largo de la guía, una vez instalado el prototipo podemos optar entre ejecutarlo con o sin funcionalidades volátiles.

Si cuenta con la imagen de Pharo "Seaside One-Click Experience 3.1", simplemente abriendo el ambiente el servidor estará activo. De lo contrario, para correr el servidor siga los siguientes pasos:

- Abrir un Workspace.
- Ejecutar el siguiente código:

```
ZnZincServerAdaptor startOn: 8080.
WAPharoServerAdapterSpecBrowser open.
```

A continuación se detallan los pasos a seguir para ambas ejecuciones:

7.4.2.1 Ejecución de la aplicación original

Si no se ha ejecutado la aplicación con las funcionalidades volátiles previamente, basta con directamente abrir la URL http://localhost:8080/store con un navegador. En caso contrario o en caso de querer asegurar la ejecución de la aplicación original, siga los siguientes pasos:

- Abrir un Workspace.
- Ejecutar el siguiente código:

```
|vfFrameworkStore|
vfFrameworkStore:=VFFrameworkStore default.
vfFrameworkStore stopScheduler.
vfFrameworkStore disableAllVFs.
```

- Abrir la URL http://localhost:8080/store con un navegador.

7.4.2.2 Ejecución de la aplicación con las funcionalidades volátiles

Para ejecutar la aplicación con las funcionalidades volátiles siga los siguientes pasos:

- Abrir un Workspace.
- Ejecutar el siguiente código:

```
|vfFrameworkStore|
vfFrameworkStore:=VFFrameworkStore default.
vfFrameworkStore startScheduler.
```

- Abrir la URL http://localhost:8080/store con un navegador.

7.5 Guía para la incorporación de funcionalidades volátiles

7.5.1 Objetivo

El objetivo de esta guía es la especificación de todos los pasos a seguir para lograr la incorporación de funcionalidades volátiles en una aplicación web desarrollada con Smalltalk y Seaside. Siguiéndola podrá lograr la implementación de funcionalidades volátiles cumpliendo con todos los conceptos definidos por el VF Framework.

7.5.2 Arquitectura desacoplada

Una aplicación web implementada con Smalltalk y Seaside generalmente se encuentra distribuida en dos paquetes. Uno correspondiente al modelo de la aplicación y otro correspondiente a la vista y al controlador de la misma (generados con Seaside). Si se desea incorporar funcionalidades volátiles mediante VF Framework, deberá crearse otro paquete que contenga todo el código correspondiente para definir las modificaciones y su gestión.

Este otro paquete contendrá clases, que extendiendo determinadas clases provistas por el Framework, actuarán como hotspots al implementar determinados métodos abstractos.

Para ello, lo primero que deberá hacer será instalar todos los archivos fuente requeridos por VF Framework. Los mismos se encuentran en el prototipo provisto.

7.5.3 Modificación de las capas de la aplicación

Las funcionalidades volátiles involucran modificaciones en las tres capas de la aplicación (capa de negocio, capa de navegación y capa de presentación).

VFFramework provee la clase “VolatileFunctionality” con el fin de representar y definir fácilmente las funcionalidades volátiles a incorporar. Esta clase debe ser extendida y cuenta con determinados métodos abstractos que actúan como hotspots. Estos métodos deben ser implementados de manera obligatoria y en caso de no ser necesario alguno de ellos se deberá dejar su cuerpo en blanco. Los métodos en cuestión se encuentran en la tabla de la Ilustración 66.

Método	Descripción
#addVars	Método para agregar variables.
#addMethods	Método para agregar métodos.
#addPointcuts	Método para agregar pointcuts.
#addMethodsModifications	Método para agregar modificaciones de métodos.
#enableSetUp	Método para agregar acciones de inicialización de activación.
#preDisable	Método para agregar acciones de preDesactivación.
#addIncludedVFs	Método para agregar funcionalidades volátiles incluidas.
#setUpScheduler	Método para agregar eventos de activación y desactivación.

Ilustración 66: Tabla con métodos de la clase "VolatileFunctionality"

Todos estos métodos a implementar y las alternativas existentes para la definición de cada uno de ellos serán explicados con más detalle en las siguientes secciones.

7.5.3.1 Modificación de capa de negocio

Las modificaciones de la capa de negocio deben modificar el comportamiento y la estructura de aquellas clases que componen el paquete con el modelo de la aplicación web.

Las funcionalidades volátiles, subclases de la clase "VolatileFunctionality", son capaces de efectuar dichas modificaciones. Para lograrlo hacen uso de un tema fundamental para el Framework como lo es la programación orientada a aspectos. Como puede observarse en el diagrama UML en la Ilustración 67, la clase "VolatileFunctionality" es a su vez subclase de "PHAspect", clase que representa un aspecto dinámico en el lenguaje de aspectos PHantom. De esta manera, resulta posible realizar en tiempo de ejecución la inyección de variables y la inyección e intercepción de métodos.

Suponga que su aplicación requiere la incorporación de una funcionalidad volátil para el lanzamiento de descuentos (DiscountVF). En tal caso, deberá definir la clase "DiscountVF" dentro de un paquete que nombraremos "Store-VFFramework". A continuación puede verse el código necesario efectuarlo:

```
VolatileFunctionality subclass: #DiscountVF
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Store-VFFramework'
```

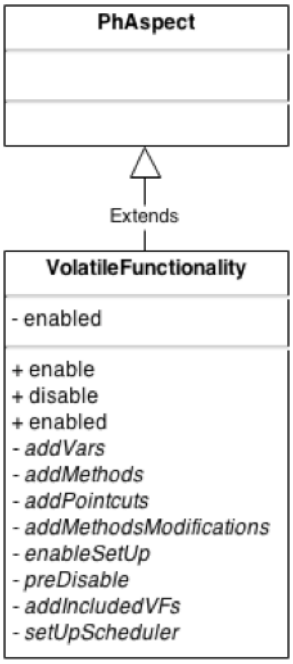


Ilustración 67: VolatileFunctionality

Inyección de variables y relaciones

Las variables y relaciones inyectadas por una funcionalidad volátil deben ser definidas dentro del método: #addVars. Para agregar variables y relaciones dentro de dicho método pueden utilizarse los métodos que han sido definidos en la tabla de la Ilustración 68.

Método	Descripción
#addInstVar: anInstVar forClass: aClass	Método para agregar la variable de instancia anInstVar a la clase aClass.
#addClassVar: aClassVar forClass: aClass	Método para agregar la variable de clase aClassVar a la clase aClass.

Ilustración 68: Tabla con métodos para inyección de variables y relaciones

De esta manera, podría agregar un atributo simplemente implementando el siguiente método:

```
addVars
        self addInstVar:'discount' forClass:'WAStoreItem'.
```

Mapeo de variables y relaciones

El mapeo de variables y relaciones no ha sido tenido en cuenta para la implementación del VF Framework en este lenguaje de programación. Esto se debe a cuestiones de tiempo y a que no fue necesario para el sample tomado como base para el desarrollo del prototipo. En lugar de utilizar una base de datos real, el mismo es manejado a partir de instancias creadas en memoria.

De cualquier manera, este punto ha sido analizado y se ha llegado a la conclusión de que el Framework podría ser extendido para cubrirlo a la perfección. La tecnología para mapeo objeto-relacional GLORP [33] y la generación de código dinámica con la que cuenta Smalltalk pareciera ser más que suficiente para conseguir dicho objetivo.

Inyección de métodos

Los nuevos métodos inyectados para una clase existente por una funcionalidad volátil deben ser definidos dentro del método: #addMethods.

Para agregar nuevos métodos dentro de dicho método pueden utilizarse los métodos definidos en la tabla de la Ilustración 69.

Método	Descripción
#addInstMethod: anInstMethod forClass: aClass	Método para agregar el método de instancia anInstMethod a la clase aClass.
#addClassMethod: aClassMethod forClass: aClass	Método para agregar el método de clase aClassMethod a la clase aClass.

Ilustración 69: Tabla con métodos para inyección de métodos

A continuación se muestra cómo pueden inyectarse getters y setters para un nuevo atributo, así también como un nuevo método:

```
addMethods

        self addInstMethod:'discount
                ^discount '
                forClass: WAStoreItem.

        self addInstMethod:'discount:aFloat
                discount:=aFloat '
                forClass: WAStoreItem.

        self addInstMethod:'addDiscount: aFloat
                discount := discount + aFloat'
                forClass: WAStoreItem.
```

Modificación del flujo original de métodos existentes

En el VF Framework desarrollado para Smalltalk existen dos alternativas a la hora de modificar el flujo original de métodos:

- **Intercepción de métodos:** Los métodos pueden ser interceptados mediante la definición de pointcuts y advices asociados. Para ello deben definirse los mismos utilizando el lenguaje de aspectos de PHantom. La definición de estas intercepciones de métodos debe ser ubicada dentro del método #addPointcuts.
- **Modificación de métodos:** Para proveer más posibilidades con el fin de alterar el comportamiento de clases del modelo, se introdujeron las

modificaciones de métodos de instancia a partir de expresiones regulares. Estas modificaciones deben ser ubicadas dentro del método #addInstMethodsModifications.

Para agregar modificaciones de métodos en una funcionalidad volátil se debe utilizar el método definido en la tabla de la Ilustración 70.

Método	Descripción
#addInstMethodModification: anInstMethodModification	Método para agregar la modificación de método de instancia anInstMethodModification.

Ilustración 70: Tabla con método para modificación de métodos

A su vez, dentro de dicho método se proveen distintas opciones para poder definir modificaciones de métodos de instancia sencillamente. Para ello, se ha optado por utilizar el patrón de diseño Strategy descripto en [20]. Este patrón permitió la definición de estrategias que deberían ser suficientes para cubrir todos los tipos de modificaciones que puedan presentarse. Tenga en cuenta que múltiples modificaciones de métodos pueden ser aplicadas de manera combinada. En la Ilustración 71, se encuentra el diagrama UML que representa el modelo de esta sección.

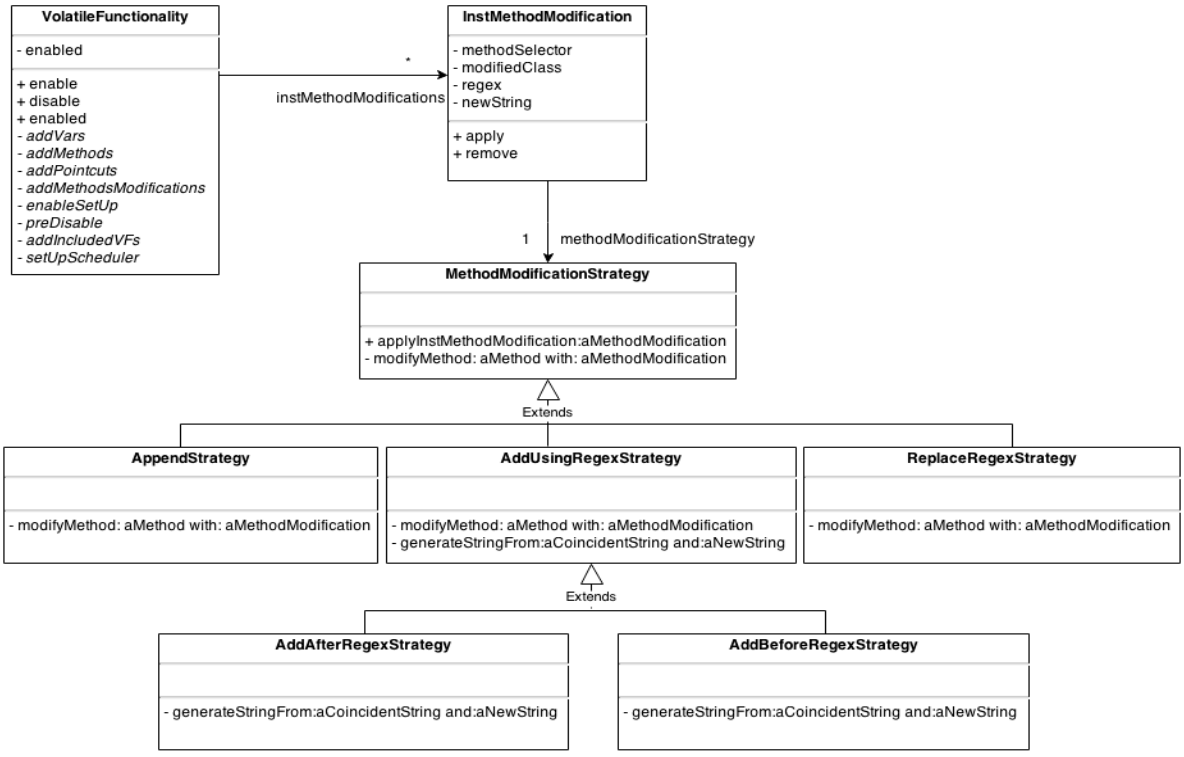


Ilustración 71: Modificaciones regex

Las modificaciones de métodos de instancia a agregar deben ser creadas a partir de los constructores definidos en la tabla de la Ilustración 72.

Método	Descripción
#instMethod: aMethodSelector forClass: aClass beforeRegex: aRegex insert: aNewString	Método para insertar la cadena de caracteres aNewString antes de la expresión regular aRegex en el método aMethodSelector de la clase aClass.
#instMethod: aMethodSelector forClass: aClass afterRegex: aRegex insert: aNewString	Método para insertar la cadena de caracteres aNewString luego de la expresión regular aRegex en el método aMethodSelector de la clase aClass.
#instMethod: aMethodSelector forClass: aClass withRegex: aRegex replacedWith: aNewString	Método para insertar la cadena de caracteres aNewString en reemplazo de la expresión regular aRegex en el método aMethodSelector de la clase aClass.
#instMethod: aMethodSelector forClass: aClass append: aNewString	Método para insertar la cadena de caracteres aNewString al final del código en el método aMethodSelector de la clase aClass.

Ilustración 72: Tabla con métodos para definir modificaciones de métodos

A continuación se muestra cómo puede modificarse un método de instancia haciendo uso de uno de los métodos descriptos:

```
addMethodsModifications

    self addInstMethodModification:
        (InstMethodModification instMethod: 'totalPrice'
         forClass: WAStoreCart
         withRegex: 'price'
        replacedWith:'finalPrice').
```

7.5.3.2 Modificación de la capa de navegación

Las modificaciones correspondientes a la capa de navegación de la aplicación para este lenguaje de programación son semejantes a las utilizadas para la modificación de la capa de negocio. Esto es así dado que Smalltalk es puramente orientado a objetos. La modificación de clases que representan los nodos y enlaces de navegación pueden requerir tanto la inyección de dependencias (semejante a la de variables y relaciones), como también la inyección, intercepción y modificación del flujo de métodos. La única diferencia radica en que las modificaciones a efectuar serán sobre clases propias de la capa de navegación que están ubicadas en el paquete de Seaside y no en el paquete del modelo de la aplicación.

7.5.3.3 Modificación de la capa de presentación

Al igual que para las modificaciones de la capa de navegación, las modificaciones de la capa de presentación también pueden modificar el comportamiento y la estructura de clases que componen el paquete de Seaside. En este caso, se hace hincapié en aquellas que definen la vista de la aplicación. Este lenguaje de programación modifica esta capa a partir de la ejecución de modificaciones en tiempo de ejecución.

Al utilizar Smalltalk y Seaside la visualización de las interfaces de una aplicación web dependen de métodos de instancia tales como #renderContentOn:html. Este tipo de métodos presentes dentro del protocolo "rendering" generan el código html para la vista de la aplicación. En la Ilustración 73 se muestra un ejemplo de uso de este método para la definición del layout de una aplicación.

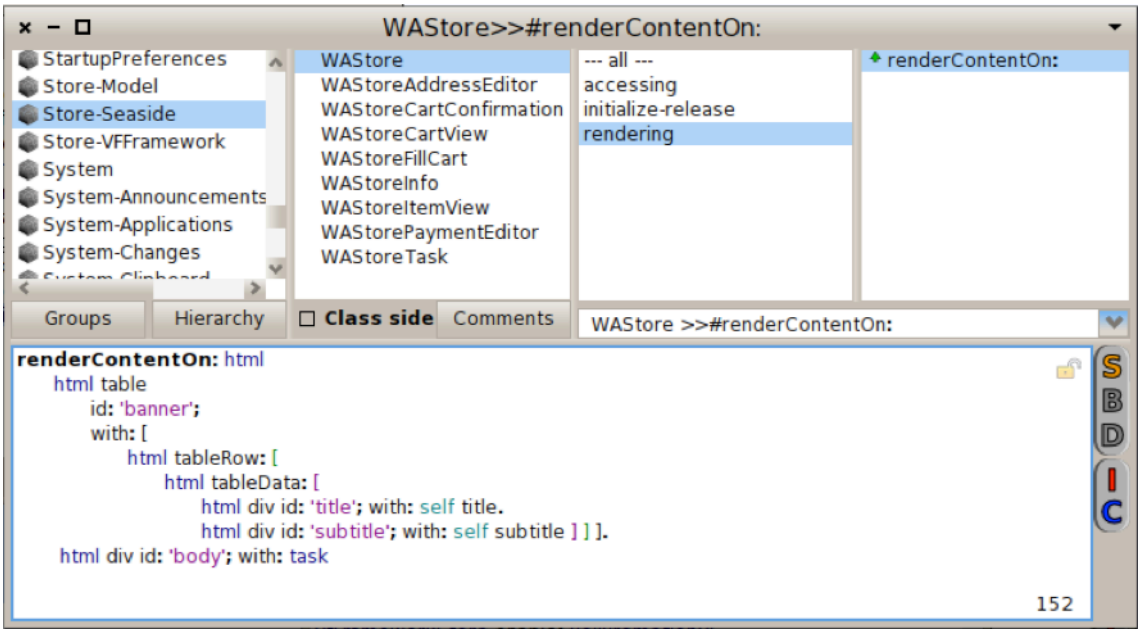


Ilustración 73: Rendering

Para la modificación de la capa de presentación puede recurrirse a los mismos mecanismos utilizados para la modificación de la capa de negocio y navegación. Sin embargo, dado que los métodos para "rendering" generalmente requieren modificaciones en secciones específicas dentro de su cuerpo, lo más adecuado suele ser la utilización de modificaciones de métodos a partir de expresiones regulares.

Solamente se ha implementado una operación extra particular para esta capa. Esta operación sirve para facilitar la adición de estilos css para la capa de presentación.

El método #style, dentro del cual se define el estilo para clases que heredan de "WAComponent" de Seaside, no siempre puede ser adicionado correctamente mediante modificaciones regex. Si este método no estuviera implementado, el mismo heredaría como código la devolución del valor "nil" y, consecuentemente, se produciría un error al aplicarle una modificación regex.

Por esta razón, se ha optado por brindar un nuevo método que tome en consideración a este punto. A partir de la definición de un pointcut y un advice asociados predeterminados pueden agregarse estilos simplemente enviando como texto el código del estilo que se desea adicionar. La definición de nuevos estilos debe

ubicarse dentro del método **#addPointcuts** haciendo uso del mensaje presente en la tabla de la Ilustración 74.

Método	Descripción
#addSeasideStyle:aStyle forClass: aClass	Método para agregar el estilo aStyle para la clase aClass.

Ilustración 74: Tabla con métodos para agregar estilos

A continuación se muestra cómo puede añadirse un nuevo estilo de esta manera:

```
addPointcuts
        self addSeasideStyle:'
                #discountedPrice { text-decoration:line-through; color:red}
                 #finalPrice { margin-left: 10px}
                #discount {color: red;font-weight: bold;font-size: 15px}'
                 forClass: 'WAStoreItemView'.
```

Nota: En caso de querer modificar un estilo existente propio de la aplicación original sí sería correcta su manipulación mediante modificaciones regex.

7.5.4 Gestión de las modificaciones

La clase "VFFramework" será capaz de controlar aquellas funcionalidades volátiles que sean añadidas. Esta clase debe ser extendida, ya que es necesaria una subclase de ella para la manipulación de las funcionalidades de una aplicación en particular. En la Ilustración 75, puede observarse el diagrama UML correspondiente a la clase "VFFramework".

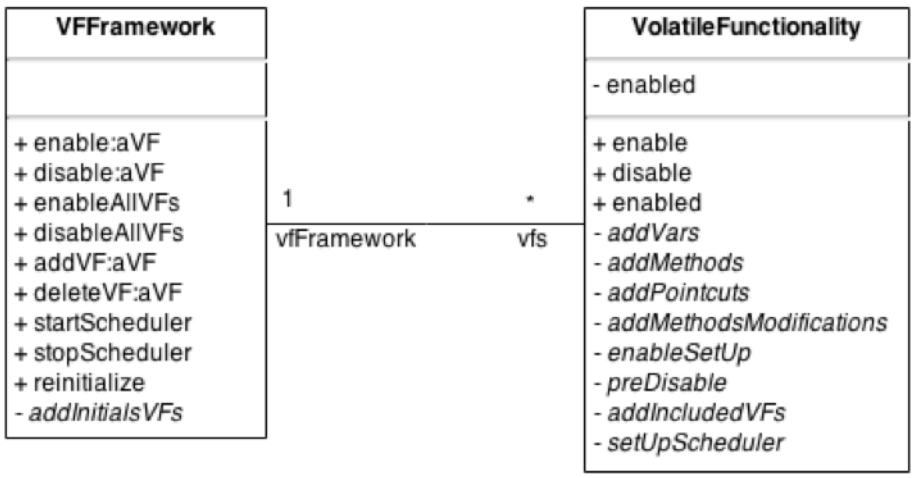


Ilustración 75: VFFramework

El único método que debe implementar esta clase de manera obligatoria es el método definido en la tabla de la Ilustración 76 para poder agregar las funcionalidades volátiles con las que el "VFFramework" en cuestión contará desde su inicio.

Método	Descripción
#addInitialVFs	Método para agregar las funcionalidades volátiles que inicialmente contendrá la aplicación.

Ilustración 76: Tabla con método para agregar FVs iniciales

A su vez, dentro de este método, se hará uso de la operación definida en la Ilustración 77 para agregar cada funcionalidad volátil deseada.

Método	Descripción
#addVF: aVFClass	Método para agregar la funcionalidad volátil cuya clase es aVFClass.

Ilustración 77: Tabla con método para agregar FVs

A continuación se muestra cómo pueden definirse ciertas funcionalidades volátiles para ser incorporadas desde el inicio en una subclase de "VFFramework":

```
addInitialVFs
        self addVF: DiscountVF.
```

En las siguientes secciones se procederá a explicar con mayor detalle todas las operaciones y decisiones de diseño correspondientes a la clase "VFFramework".

7.5.4.1 Administración de las funcionalidades volátiles

La clase "VFFramework" es la encargada de gestionar y administrar todas las funcionalidades volátiles de la aplicación. Para ello cuenta con una importante cantidad de operaciones que se encuentran definidas en la tabla de la Ilustración 78. Todas estas operaciones pueden ser invocadas directamente desde el workspace.

Método	Descripción
#enable:aVF	Método para activar la funcionalidad volátil aVF.
#disable:aVF	Método para desactivar la funcionalidad volátil aVF.
#enableAllVFs	Método para activar todas las funcionalidades volátiles incorporadas.
#disableAllVFs	Método para desactivar todas las funcionalidades volátiles incorporadas.
#addVF:aVF	Método para agregar la funcionalidad volátil aVF.
#deleteVF:aVF	Método para remover la funcionalidad volátil aVF.
#startScheduler	Método para iniciar la ejecución de eventos de activación y desactivación programados.

#stopScheduler	Método para parar la ejecución de eventos de activación y desactivación programados.
# reInitialize	Método para parar la ejecución de eventos, desactivar todas las funcionalidades volátiles e inicializar nuevamente la subclase de VFFramework. Este método es útil especialmente para cuando el código de alguna funcionalidad volátil es modificado. Sin una reinicialización las modificaciones realizadas no serán reflejadas.

Ilustración 78: Tabla con métodos para administración de FVs

Para controlar efectivamente la concurrencia de las operaciones principales del VFFramework se ha utilizado el patrón de diseño Singleton y un semáforo de exclusión mutua. Así, sólo existe una instancia de la subclase de "VFFramework" y las operaciones más importantes están protegidas de manera tal que sólo un hilo pueda ser ejecutado a la vez. En la Ilustración 79, puede observarse el diagrama UML que representa dicha situación.

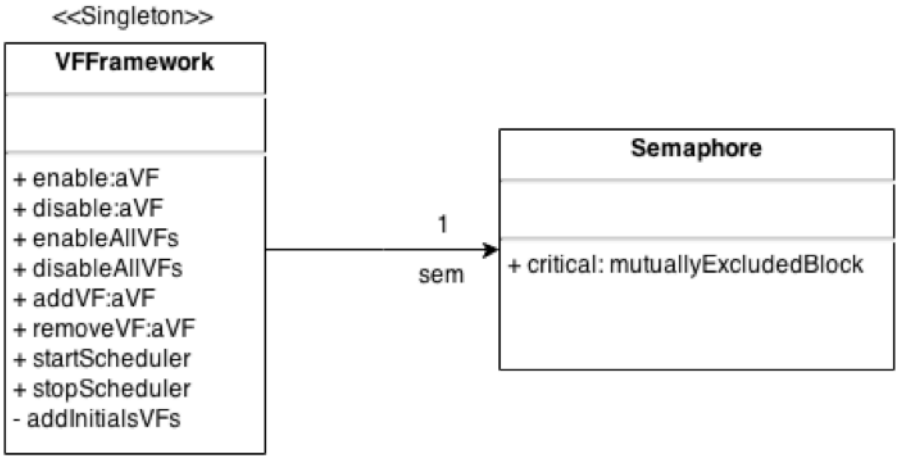


Ilustración 79: Concurrencia de VFFramework

7.5.4.2 Weaving de las funcionalidades volátiles

Debido a la potencia y el tipado dinámico que Smalltalk posee, se ha utilizado weaving dinámico. Esto posibilita la integración dinámica de las funcionalidades volátiles. Efectivamente, así las funcionalidades volátiles pueden ser activadas, desactivadas, incorporadas y removidas en tiempo de ejecución. Ante cualquiera de estas operaciones deberá ejecutarse el weaving correspondiente. La subclase de VFFramework es la encargada de coordinar este procedimiento de manera transparente para quien hace uso de este Framework.

7.5.4.3 Subetapas de las funcionalidades volátiles

Ambas subetapas de las funcionalidades volátiles cuentan con métodos dentro de los cuales deben ser definidas las acciones a ejecutar. De no existir, deben quedar vacíos. En este caso, no se brindarán operaciones predefinidas para utilizar dentro. Esto se debe a que las acciones a ejecutar en estos puntos son totalmente independientes y pueden variar según el caso particular.

Inicialización de activación: Para programar las acciones a ejecutar en el instante posterior a que una funcionalidad volátil es activada, debe definirse dentro del método #enableSetUp.

PreDesactivación: Para programar las acciones a ejecutar en el instante previo a que una funcionalidad volátil es desactivada, debe definirse dentro del método #preDisable.

A continuación se muestra un ejemplo de uso de una definición de una inicialización de activación:

```
enableSetUp
        (WAStoreInventory default) allItems do:[:item | item discount:0.].
```

7.5.4.4 Commons para funcionalidades volátiles

Los funcionalidades volátiles pueden contener otras funcionalidades volátiles incluidas como "Commons", para lo cual deben definirlas dentro del método #addIncludedVFs. Esta relación se encuentra representada en el diagrama UML de la Ilustración 80.

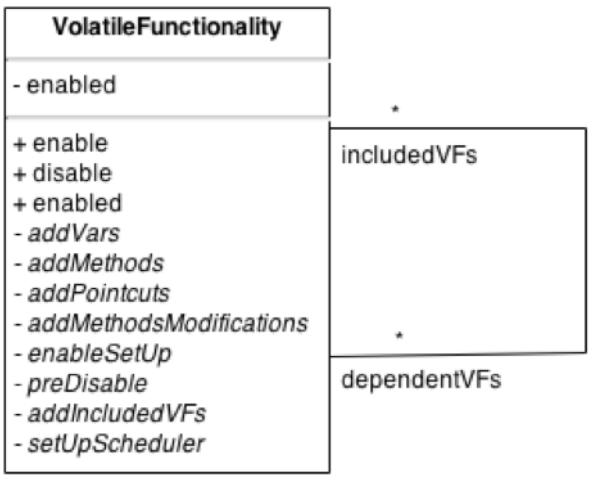


Ilustración 80: Commons de VolatileFunctionality

Para agregar funcionalidades volátiles como "Commons" dentro de dicho método debe utilizarse la operación que se encuentra definida en la tabla de la Ilustración 81.

Método	Descripción
#addIncludedVF: aVFClass	Método para incluir la funcionalidad volátil de clase aVFClass.

Ilustración 81: Tabla con método para agregar FVs como "Common"

Esta operación se encarga no sólo de agregar una funcionalidad volátil como "Common", sino también de validar que sólo exista una única instancia de dicha funcionalidad. Este punto es importante, ya que una funcionalidad volátil que es compartida debe tener referencia de las funcionalidades que dependen de ella. De esta manera, puede manipularse la activación y desactivación de las funcionalidades correctamente.

A continuación se muestra cómo puede agregarse una funcionalidad volátil como "Common":

```
addIncludedVFs
        self addIncludedVF: DiscountVF.
```

7.5.4.5 Programación de eventos de activación y desactivación

La programación de eventos de activación y desactivación no ha sido implementada mediante un motor de reglas en este caso. No sólo no se ha encontrado un buen motor de reglas para este lenguaje de programación, sino que también pareció correcta la oportunidad para mostrar otra alternativa diferente de implementación.

Se ha llegado a una solución capaz de controlar esta cuestión mediante la combinación de un Scheduler y aspectos (en un lenguaje dinámicamente tipado).

Las funcionalidades volátiles cuentan con una colección de eventos de activación y una colección de eventos de desactivación. Estos eventos son instancias de la clase "EventAspect". "EventAspect" es a su vez subclase de "PHAspect" (clase que representa un aspecto dinámico en Phantom) y por lo tanto sus instancias pueden ser instaladas o desinstaladas en tiempo de ejecución.

Internamente, los eventos temporales son manejados por el Scheduler, los eventos de negocio con aspectos (instancias de la clase "EventAspect") y los eventos de negocio temporales por una combinación de ambos. Un evento de negocio temporal utiliza el Scheduler para definir el momento a partir del cual se considerará determinado evento de negocio. Dicho evento de negocio no será otra cosa sino una instancia de la clase "EventAspect".

Los eventos de activación y desactivación sólo serán tenidos en cuenta en caso de que el Scheduler de "VFFramework" haya sido iniciado. En ese caso, las funcionalidades volátiles desactivadas considerarán los eventos de activación y las funcionalidades volátiles activadas considerarán los eventos de desactivación.

En la Ilustración 82, se encuentra el diagrama UML con todas las clases involucradas representadas.

Las funcionalidades volátiles deben definir sus eventos programados dentro del método #schedule. Para programarlos cuentan con una amplia variedad de operaciones. El manejo interno de estas operaciones es transparente al programador.

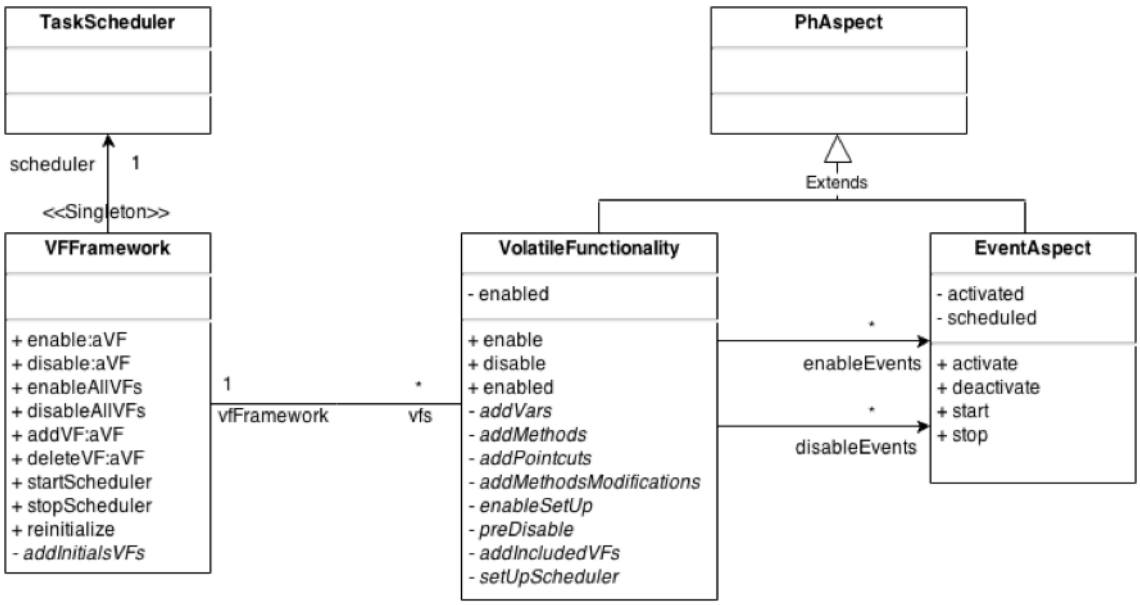


Ilustración 82: Eventos de activación y desactivación

En la tabla de la Ilustración 83 se encuentran definidos los métodos para la programación de eventos temporales.

Método	Descripción
#scheduleEnableAt: when	Método para programar la activación para el momento when.
#scheduleEnableAt: when every: aDuration	Método para programar la activación por primera vez para el momento when y luego cada una duración aDuration.
#scheduleEnableEvery: aDuration	Método para programar la activación cada una duración aDuration.
#scheduleEnableOnceAt: when	Método para programar la activación por única vez para el momento when.
#scheduleDisableAfter:aDuration	Método para programar la desactivación, luego de una duración aDuration, luego de haber sido activada.
#scheduleDisableAt: when	Método para programar la desactivación para el momento when.
#scheduleDisableAt: when every: aDuration	Método para programar la desactivación por primera vez para el momento when y luego cada una duración aDuration.
#scheduleDisableEvery: aDuration	Método para programar la desactivación cada una duración aDuration.
#scheduleDisableOnceAt: when	Método para programar la desactivación por única vez para el momento when.

Ilustración 83: Tabla con método para definir eventos temporales

En la tabla de la Ilustración 84 se encuentran definidos los métodos para la programación de eventos de negocio.

Método	Descripción
#scheduleEnableEventReceiver: aClass selector: aSelector	Método para programar la activación para cuando se llame al método aSelector de la clase aClass.
#scheduleEnableEventReceiver: aClass selector: aSelector condition: aCondition	Método para programar la activación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass.
#scheduleDisableEventReceiver: aClass selector: aSelector	Método para programar la desactivación para cuando se llame al método aSelector de la clase aClass.
#scheduleDisableEventReceiver: aClass selector: aSelector condition: aCondition	Método para programar la desactivación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass.

Ilustración 84: Tabla con método para definir eventos de negocio

En la tabla de la Ilustración 85 se encuentran definidos los métodos para la programación de eventos de negocio temporales.

Método	Descripción
#scheduleEnableEventReceiver: aClass selector: aSelector condition: aCondition at: when	Método para programar la activación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta a partir del momento when.
#scheduleEnableEventReceiver: aClass selector: aSelector condition: aCondition at: when every: aDuration	Método para programar la activación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta por primera vez para el momento when y luego cada una duración aDuration.
#scheduleEnableEventReceiver: aClass selector: aSelector condition: aCondition every: aDuration	Método para programar la activación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta cada una duración aDuration.
#scheduleEnableEventReceiver: aClass selector: aSelector condition: aCondition onceAt: when	Método para programar la activación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta por única vez para el momento when.
#scheduleDisableEventReceiver: aClass selector: aSelector condition: aCondition at: when	Método para programar la desactivación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta para el momento when.
#scheduleDisableEventReceiver: aClass selector: aSelector condition: aCondition	Método para programar la desactivación para cuando se cumpla la condición

at: when every: aDuration	aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta por primera vez para el momento when y luego cada una duración aDuration.
#scheduleDisableEventReceiver: aClass selector: aSelector condition: aCondition every: aDuration	Método para programar la desactivación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta cada una duración aDuration.
#scheduleDisableEventReceiver: aClass selector:aSelector condition:aCondition onceAt: when	Método para programar la desactivación para cuando se cumpla la condición aCondition al llamar al método aSelector de la clase aClass. Es tenido en cuenta por única vez para el momento when.

Ilustración 85: Tabla con método para definir eventos de negocio temporales

A continuación se muestra un ejemplo de cómo puede definirse un evento de activación:

```
setUpScheduler
        self scheduleEnableAt: (DateAndTime now + 1 minute)
                every: 10 minutes.
```

Nota: Como ya ha sido indicado, "VFFramework" posee las operaciones #startScheduler y #stopScheduler para iniciar o detener la ejecución de los eventos programados.

7.5.4.6 Ejecución

Si ha descargado la imagen "Seaside One-Click Experience 3.1", simplemente abriendo el ambiente el servidor estará activo. De lo contrario, para correr el servidor siga los siguientes pasos:

- Abrir un Workspace.
- Ejecutar el siguiente código:

```
ZnZincServerAdaptor startOn: 8080.
WAPharoServerAdapterSpecBrowser open.
```

Como fue explicado, podemos optar entre ejecutar la aplicación web con o sin funcionalidades volátiles.

A continuación se detallan los pasos a seguir para ambas ejecuciones.

7.5.4.6.1 Ejecución de la aplicación original

Si no se ha ejecutado la aplicación con las funcionalidades volátiles previamente, basta con directamente abrir la URL http://localhost:8080/**appWeb** con un navegador. En caso contrario o en caso de querer asegurar la ejecución de la aplicación original, siga los siguientes pasos:

- Abrir un Workspace.
- Ejecutar el siguiente código:

```
|vfFramework|
vfFramework:=VFFrameworkAppWeb default.
vfFramework disableAllVFs.
```

- Abrir la URL http://localhost:8080/**appWeb** con un navegador.

Nota: Tanto la clase VFFramework**AppWeb** como la URL http://localhost:8080/**appWeb** deberían ser reemplazados por el nombre correspondiente a la aplicación que se desea correr.

7.5.4.6.2 Ejecución de la aplicación con las funcionalidades volátiles

Para ejecutar la aplicación con las funcionalidades volátiles siga los siguientes pasos:

- Abrir un Workspace.
- Ejecutar el siguiente código:

```
|vfFramework|
vfFramework:= VFFrameworkAppWeb default.
vfFramework startScheduler.
```

- Abrir la URL http://localhost:8080/**appWeb** con un navegador.

Nota: Tanto la clase VFFramework**AppWeb** como la URL http://localhost:8080/**appWeb** deberían ser reemplazados por el nombre correspondiente a la aplicación que se desea correr.

8 Comparación entre prototipos implementados

8.1 Tabla comparativa

En los capítulos 6 y 7 se han presentado los prototipos de VF Framework desarrollados para los lenguajes de programación Java y Smalltalk. En este capítulo se procederá a comparar ambas implementaciones. Serán contrastadas indicando de manera resumida qué cuestiones fueron cubiertas y qué estrategias o tecnologías fueron utilizadas.

En la tabla de la Ilustración 86 se presenta una tabla comparativa entre los dos prototipos implementados.

Categoría	Subcategoría	Prototipo Java	Prototipo Smalltalk
Arquitectura desacoplada	-	SI	SI
Modificación de capa de negocio	Inyección de variables y relaciones	SI (Aspectos)	SI (Aspectos)
Modificación de capa de negocio	Mapeo de variables y relaciones	SI (Anotaciones)	NO (**)
Modificación de capa de negocio	Inyección de métodos	SI (Aspectos)	SI (Aspectos)
Modificación de capa de negocio	Modificación del flujo original de métodos	SI (Aspectos)	SI (Aspectos y modificaciones Regex)
Modificación de capa de navegación	-	SI (Aspectos)	SI (Aspectos y modificaciones Regex)
Modificación de capa de presentación	-	SI (XSLTs)	SI (Aspectos y modificaciones Regex)
Administración centralizada de las funcionalidades volátiles	-	SI (Servicio Singleton con operaciones principales definidas como synchronized)	SI (Clase Singleton con semáforo de exclusión mutua para operaciones principales)
Weaving	-	SI (Estático con Maven y XSLT)	SI (Dinámico con generación dinámica de código)
Subetapas	-	NO (*)	SI
Commons	-	NO (**)	SI
Eventos de	Eventos temporales	SI	SI

activación y desactivación		(Motor de reglas Drools)	(Scheduler)
Eventos de activación y desactivación	Eventos de negocio	NO (*) (Motor de reglas Drools)	SI (Aspectos dinámicos)
Eventos de activación y desactivación	Eventos de negocio temporales	NO (*) (Motor de reglas Drools)	SI (Aspectos dinámicos + Scheduler)

Ilustración 86: Tabla comparativa entre los dos prototipos implementados

(*) Esta cuestión no ha sido incluida en el prototipo, pero está cubierta y puede ser implementada.

(**) Esta cuestión no ha sido incluida en el prototipo, ni está cubierta, pero podría ser implementada.

8.2 Detalle de comparación

Arquitectura desacoplada

Ambos prototipos implementan una arquitectura desacoplada en la que por un lado se encuentra el proyecto original (sin modificaciones) y por otro lado se encuentra el proyecto correspondiente a las funcionalidades volátiles. Sin embargo, existe una gran diferencia entre ambos prototipos.

En el prototipo en Smalltalk, si bien se han incluído todas las funcionalidades volátiles en el mismo proyecto, si se deseara se podrían incluir sólo un subconjunto de ellas. Cada una de las funcionalidades volátiles se encuentra desacoplada de las demás y contiene en su implementación todos sus cambios a efectuar en la aplicación.

En el prototipo en Java, en cambio, los cambios pertenecientes a las funcionalidades volátiles no se encuentran contenidos dentro de una sola clase y desacoplados entre sí. Por el contrario, están dispersos y presentes conjuntamente en múltiples sectores dentro del proyecto. Por supuesto, es posible y tendría sentido que existieran múltiples proyectos desacoplados (uno por cada funcionalidad volátil) y que los mismos pudieran ser encadenados con el propósito de correr una aplicación con las funcionalidades volátiles deseadas. Sin embargo, por cuestiones de tiempo, para este prototipo esto no ha sido realizado.

Modificación de la capa de negocio

Ambos prototipos son capaces de modificar la capa de negocio mediante la utilización de aspectos. Esta tecnología basta para lograr la inyección de variables y métodos, así como la modificación del flujo original de métodos existentes a partir de la intercepción de métodos. No obstante, la manera de la que los aspectos son utilizados varía dado que Java es estáticamente tipado y Smalltalk es dinámicamente tipado. Por esta razón, Smalltalk cuenta con la posibilidad adicional de utilizar modificaciones regex para realizar modificaciones de la capa de negocio. Otra gran diferencia que poseen los prototipos presentados radica en la manera en que se decide cuándo ejecutar los métodos con sus flujos originales y cuándo ejecutarlos con sus flujos modificados. El prototipo en Java, requiere de sentencias condicionales que basadas en el estado de cierta funcionalidad volátil definen esta cuestión. Esto se debe a que al ser estático, el código fuente compilado siempre será el mismo y no podrá ser modificado. El prototipo

en Smalltalk, en cambio, sólo posee el código a ejecutar, sin la necesidad de utilizar sentencias condicionales de ningún tipo. Esto se debe a que al ser dinámico, el código fuente presente en la aplicación para un momento dado siempre será el adecuado para el estado de la funcionalidad volátil pertinente. Cada vez que una funcionalidad volátil es activada o desactivada, se efectúan las modificaciones correspondientes en sus métodos involucrados.

En cuanto al mapeo de variables inyectadas, el prototipo para Java lo ha cubierto perfectamente. El hecho de que las entidades estén mapeadas mediante anotaciones, hace que al inyectar las variables, estas puedan ser mapeadas de manera simple y directa. En el prototipo para Smalltalk, en cambio, el mapeo de variables no ha sido tenido en cuenta. Este último ha utilizado un sample que no utiliza una base de datos real, sino que es manejado a partir de instancias creadas en memoria. No obstante, se ha analizado y debería poder ser cubierto perfectamente mediante la utilización de la tecnología para mapeo objeto-relacional GLORP.

Modificación de la capa de navegación

Ambos prototipos son capaces de modificar la capa de navegación y lo hacen utilizando los mismos mecanismos que para modificar la capa de negocio. Aspectos en Java y aspectos y modificaciones regex en Smalltalk, con las ya mencionadas diferencias entre sí.

Modificación de la capa de presentación

Las tecnologías utilizadas para la modificación de la capa de presentación de los dos prototipos son muy diferentes, y lo mismo ocurre con las estrategias utilizadas. En Java se utilizan transformaciones XSLT de documentos XHTML. Los documentos son transformados en tiempo de compilación utilizando la estrategia de “visualización condicional”. En Smalltalk en cambio, dado que la capa de presentación depende de objetos, se pueden utilizar los mismos mecanismos que para las modificaciones de las capas de negocio y navegación. Lo más conveniente, sin embargo, suele ser la utilización de modificaciones regex. La estrategia utilizada es la de “ejecución en tiempo de compilación”.

La modificación de la capa de presentación en Smalltalk resulta una solución mucho más sencilla, potente y escalable que en Java. Por un lado, las modificaciones regex son fáciles de definir y cuentan con las operaciones preestablecidas del VFFramework para hacerlo. Además, las modificaciones a incorporar en Smalltalk no requieren de ningún tipo de lógica condicional, lo cual las vuelve aún más simples. Las transformaciones XSLT de Java, por otro lado, son más complejas y pueden resultar un tanto engorrosas a la hora de implementar. A su vez, esta mayor complejidad también puede traer aparejadas mayores dificultades en cuanto a la escalabilidad de las modificaciones.

Administración centralizada de las funcionalidades volátiles

Ambos prototipos cuentan con una administración centralizada de las funcionalidades volátiles. Es decir, cuentan con un punto de entrada único mediante el cual se puede acceder a las operaciones principales para el manejo de las funcionalidades volátiles. Para ello, en ambos casos se utilizó el patrón de diseño Singleton para que exista una única instancia. Además, en ambos casos se hizo especial hincapié en el control de la concurrencia. En Java esto fue posible mediante métodos synchronized y en Smalltalk mediante un semáforo de exclusión mutua.

Weaving

El weaving necesario para la integración del proyecto original y de las funcionalidades volátiles ha podido ser llevado a cabo sin problemas en ambos casos. Para el prototipo en Java se efectuó un weaving estático utilizando Maven durante la fase de compilación, sumado a una transformación XSL del archivo de configuración de la aplicación original. Para el prototipo en Smalltalk se efectuó un weaving dinámico utilizando la generación dinámica de código propia de Smalltalk.

Subetapas

En el prototipo para Java no se han definido subetapas por falta de necesidad. No obstante, este concepto puede ser implementado perfectamente y así ha sido explicado en la sección correspondiente. En el prototipo para Smalltalk, este punto ha sido cubierto en su totalidad y se han definido múltiples casos de "Inicialización de activación" y "PreDesactivación".

Commons

Esta cuestión ha sido cubierta en el prototipo para Smalltalk, pero no ha sido tenida en cuenta, ni requerida para el prototipo para Java. No obstante, las relaciones y consideraciones necesarias para la implementación de dicho punto podrían ser establecidas sin ningún inconveniente. La implementación y la lógica necesaria sería muy similar a la utilizada para Smalltalk.

Programación de eventos de activación y desactivación

Este punto pudo ser cubierto satisfactoriamente para ambos prototipos. Se logró una correcta programación de eventos. Para el prototipo en Java se utilizó el motor de reglas Drools, mientras que para el prototipo en Smalltalk no se utilizó un motor de reglas, pero se llegó a un funcionamiento similar a partir del uso de aspectos y un Scheduler.

8.3 Conclusiones

La implementación de prototipos utilizando lenguajes de programación diferentes como lo son Java y Smalltalk fue muy relevante para el desarrollo de esta tesis de grado. Estos prototipos permitieron no sólo poner en práctica los conceptos de VF Framework, sino también considerar diferentes alternativas y estrategias para su implementación. Gracias a ello se obtuvo una mejor y más amplia visión que ayudó a definir los requerimientos fundamentales con los que debe contar un lenguaje de programación para hacer uso de VF Framework.

Si bien ambos prototipos cubrieron satisfactoriamente los conceptos más importantes de VF Framework (y podrían cubrir los faltantes), se ha obtenido un mejor resultado con el prototipo de Smalltalk. A diferencia de Java, Smalltalk es un lenguaje dinámicamente tipado, lo cual lo vuelve una alternativa muy potente y de fácil manipulación. Estas características tienen un gran impacto, ya que posibilitan una implementación mucho más dinámica y con más y mejores opciones para poder llevarlo a cabo.

Además, para Smalltalk se ha desarrollado un Framework de caja blanca que facilita en gran medida el proceso de implementación de funcionalidades volátiles según los conceptos de VF Framework. Simplemente a partir de la extensión de determinadas clases y de la implementación de determinados hotspots, resulta posible la incorporación desacoplada de funcionalidades volátiles y la programación de eventos de

activación y desactivación para las mismas. Esta tarea resulta más costosa para el caso de Java, donde las modificaciones y programaciones de eventos correspondientes a las distintas funcionalidades volátiles se encuentran distribuidas sin una clara separación entre sí.

9 Experimento

9.1 Descripción

Se ha realizado un experimento, cuyo principal objetivo ha sido la obtención de feed-back de desarrolladores imparciales acerca del marco de trabajo desarrollado.

Para comenzar, se les presentó brevemente a los participantes del experimento el VF Framework y los principales conceptos que conlleva. Esta presentación constó de una explicación oral de aproximadamente 40 minutos, acompañada por una serie de diapositivas proyectadas.

Luego, los participantes recibieron una guía.

Con el propósito de facilitar y agilizar la puesta a punto del experimento, se utilizó el software para virtualización de arquitecturas VirtualBox. El mismo fue empleado para correr la imagen de una máquina virtual que poseía todo lo necesario para la realización del experimento. Esta imagen incluía el prototipo de VF Framework realizado para Java, que ha sido presentado en la sección 6.4. Dicho prototipo de la aplicación booking-faces cuenta con las funcionalidades volátiles incorporadas según los conceptos de VF Framework.

Dentro de la guía se incluyeron las instrucciones para efectuar la instalación de la imagen de la máquina virtual. Además, se entregó una descripción de la aplicación web original y de las funcionalidades volátiles añadidas en el prototipo.

Por último, se detallaron los pasos a seguir y se solicitó la ejecución y prueba del prototipo, primero con las funcionalidades volátiles incorporadas y luego sin ellas.

La idea de este experimento apuntó a brindarles a los participantes la posibilidad de ejecutar, probar y recorrer por su cuenta uno de los prototipos implementados. De esta manera, se les permitió visualizar un ejemplo concreto que pudiera sostener aquellos conceptos teóricos que les habían sido presentados previamente.

Una vez finalizada esta tarea, nos interesó conocer el nivel de entendimiento, el nivel de aceptación, la calidad de la experiencia y las opiniones de los participantes acerca del VF Framework. Por esta razón, se les indicó que completaran una encuesta anónima con las siguientes preguntas:

- Cómo detectó dónde realizar los cambios para eliminar una funcionalidad volátil?
- Cuanto tiempo le llevó detectar donde modificar ? (Components, classes, resources, etc)
- A medida que realizaba los cambios, introdujo errores que posterirmente debió corregir? **No – Muy pocas veces – Normalmente – Siempre**
- Cuánto tiempo le llevó realmente remover todas las funcionalidades volátiles?
- Cuántos ciclos de testing debió realizar para obtener una versión estable (sin errores) de la aplicación? **1 – 2 – 3 – 5 – 10 – Más**
- Siente que la aplicación es confiable? (Es decir que puede ser utilizada sin errores) **Si – No**
- Recomendaría que es necesario realizar mas pruebas? **Si – No**
- En términos de estabilidad, este enfoque es **Menos estable – Igual de estable – Más estable**

- En términos de complejidad, este enfoque es **Menos estable – Igual de estable – Más estable**
- En términos de mantenibilidad, este enfoque es **Menos estable – Igual de estable – Más estable**
- En la aplicación booking-faces, cómo haría para modificar el banner de la promoción del long weekend?
- Describa brevemente cómo haría para remover las funcionalidades volátiles con este enfoque?
- Qué opinión tiene sobre el enfoque presentado?
- Qué ventajas y qué desventajas observa?

Dado que el experimento pretendía recoger un feed-back imparcial, sólo se respondieron dudas respecto al significado de las preguntas. Se les explicó a los encuestados que el resto de las consultas no podrían ser respondidas (hasta finalizar el experimento) para evitar el condicionamiento de sus respuestas finales.

9.2 Participantes

Participaron del experimento:

- Alumnos de la materia "Diseño de aplicaciones web", correspondiente al cuarto año de la carrera "Licenciatura en Sistema" de la Facultad de Informática de la UNLP.
- Alumnos del curso de capacitación del LIFIA (Laboratorio de Investigación y Formación en Informática Avanzada) de la Facultad de Informática de la UNLP.

Hasta el momento se han tomado 14 muestras.

9.3 Análisis y evaluación de los resultados del experimento

Teniendo en cuenta la encuesta realizada, podemos destacar que un 80% de los participantes respondieron que sentían que la aplicación era confiable.

Luego, comparando el enfoque presentado con el enfoque tradicional ad-hoc para la manipulación de funcionalidades volátiles, se puede destacar que:

- En términos de estabilidad, el 73% aseguró que era más estable, mientras que el 20% respondió que era igual de estable.
- En términos de complejidad, el 67% aseguró que era igual de estable y el 20% contestó que era más estable.
- En términos de mantenibilidad, el 73% sostuvo que era más estable y el 20% igual de estable.

Algunas de las preguntas incluidas han tenido un propósito técnico, cuyo objetivo ha sido medir el porcentaje de participantes que ha comprendido el enfoque correctamente. Tras analizar sus respuestas, se puede asegurar que un 85,71% ha sido exitoso en ello.

Finalmente, con relación a las opiniones de los encuestados, podemos señalar que todos ellos han aceptado este enfoque de manera positiva. La mayoría de ellos ha hecho énfasis en las ventajas, tales como la mayor simplicidad y facilidad para la etapa de mantenimiento, la preservación de la integridad de la aplicación original, y la posibilidad extra de poder programar la activación o desactivación de las funcionalidades volátiles mediante eventos. Sin embargo, algunos de ellos también han expresado que aunque este enfoque pueda ser muy útil para el mantenimiento de una aplicación, a la vez puede resultar bastante complejo, y puede requerir un alto nivel de conocimiento para implementar funcionalidades volátiles de esta manera.

9.4 Conclusiones

En base a los resultados obtenidos del experimento, se puede concluir que el enfoque ha tenido una muy buena aceptación por parte de los desarrolladores que han participado. La mayoría de ellos lo ha comprendido correctamente y lo ha calificado de manera positiva, resaltando las mejoras y la conveniencia que presenta con respecto a una implementación tradicional. Por supuesto, esto siempre y cuando el mismo sea aplicado a situaciones que ameriten el uso de funcionalidades volátiles.

Cabe destacar que una vez terminado el experimento, algunos de los participantes se mostraron interesados en el marco de trabajo presentado y hasta se acercaron a realizar diversas consultas sobre el mismo. Lógicamente, se agradeció su interés y, ahora sí, todas sus preguntas fueron respondidas.

10 Trabajos futuros

En este capítulo se definirán los trabajos futuros a efectuar con el fin de continuar y avanzar con la investigación realizada. Los mismos serán clasificados según las posibles ramas de investigación.

10.1 Estudios y evaluaciones de impacto

A lo largo de la tesis, se ha hecho énfasis en los problemas que la aparición de funcionalidades volátiles puede provocar en aplicaciones web desarrolladas de manera ad-hoc o tradicional.

Entre ellos se han mencionado la pérdida de calidad del código, la introducción de errores y la necesidad de efectuar múltiples ciclos de testing.

Estas cuestiones también tienen impacto sobre los desarrolladores, ya que conllevan tiempo y esfuerzo de trabajo extra por parte de ellos.

A su vez, los clientes también se ven afectados por estos problemas. Al fin y al cabo, son ellos quienes sufren el aumento de sus costos y obtienen un producto de menor calidad y de manera no inmediata.

Teniendo en cuenta todos estos factores, resulta claro que contar con un marco de trabajo como VF Framework podría ser muy beneficioso. Sin embargo, sería ideal y restaría para la investigación contar con datos obtenidos de la realidad que lo sustenten. Para ello, se requeriría la realización de estudios y evaluaciones de impacto que permitieran medir empíricamente las cuestiones descriptas, y posteriormente comparar lo que ocurre al utilizar el framework propuesto.

A partir de esta investigación se podrían obtener datos para respaldar el desarrollo realizado. Además, en el futuro estos datos podrían resultar muy útiles para estimar cuestiones tales como el retorno sobre la inversión para entidades que hagan uso de este enfoque. Con estos datos sería posible calcular aproximadamente cuanto se ahorraría utilizándolo para determinado proyecto.

10.2 Avance de desarrollo

El futuro del enfoque básicamente consta de profundizar cada uno de los conceptos para obtener mejores resultados.

A pesar de que la idea general está clara, la implementación concreta de la misma aún necesita ser pulida.

Hasta el momento, se han desarrollado dos prototipos con un propósito meramente experimental con un saldo muy positivo. Estos prototipos permitieron no sólo poner en práctica los conceptos de VF Framework, sino también considerar diferentes alternativas y estrategias para la implementación. Gracias a ello se obtuvo una mejor y más amplia visión que ayudó a definir los requerimientos fundamentales con los que debe contar un lenguaje de programación para hacer uso de VF Framework.

No obstante, estos prototipos deberían seguir siendo mejorados, optimizando su funcionamiento, facilitando su implementación y cubriendo algunos puntos que han sido dejados de lado por cuestiones de tiempo. Inclusive, se deberían seguir desarrollando nuevos prototipos, utilizando nuevas tecnologías y nuevos lenguajes de programación.

Al consolidarse implementaciones sólidas y maduras de VF Framework para los lenguajes de programación estudiados, el siguiente paso de la investigación podría

apuntar a la creación de Frameworks similares al desarrollado para el prototipo de Smalltalk. Es decir, Frameworks que a partir de la extensión de determinadas clases e implementación de determinados métodos, faciliten aún más la construcción de una aplicación web con funcionalidades volátiles. Este punto resultaría vital para que el enfoque pudiera posicionarse y comenzar a ser utilizado en la industria por la comunidad.

Por último, actualmente el Framework no considera un entorno gráfico para la configuración de las reglas de activación y desactivación de las funcionalidades volátiles. Por el contrario, utiliza mecanismos que sólo pueden ser implementados por quienes se encargan de desarrollar la aplicación. No obstante, podría considerarse la utilización de un entorno como Guvnor-Drools [16]. De esta manera, un usuario sin grandes conocimientos informáticos podría ser capaz de definir sus propias reglas de activación y desactivación para las funcionalidades.

11 Conclusiones

Como conclusiones finales de esta tesis, pueden destacarse algunas cuestiones interesantes que se han obtenido a lo largo del desarrollo de la misma.

En primer lugar, tras la investigación realizada, resulta clara la necesidad actual de contar con alguna alternativa que permita enfrentar eficazmente la constante volatilidad y dinamismo creciente que presentan las aplicaciones web hoy en día. Para ello, es imprescindible poder mejorar la manipulación de aplicaciones web ante la aparición de funcionalidades volátiles. Las mismas representan un gran problema durante la etapa de mantenimiento de una aplicación. Cuestiones tales como la calidad del código resultante, los tiempos requeridos, y los costos se ven directamente perjudicados.

El Framework propuesto justamente provee soluciones para mejorar el ciclo de vida de las funcionalidades volátiles en aplicaciones web. Esto en efecto, permite mejorar la etapa de mantenimiento de aplicaciones web en las cuales la aparición de funcionalidades volátiles son una cuestión recurrente. Como resultado, la calidad del código no se degrada, se preserva la integridad tanto de la aplicación original como de las funcionalidades volátiles en sí, y se reducen los tiempos y los costos. Además, su adopción puede proporcionar grandes ventajas, especialmente para aquellas aplicaciones en las que también se conocen con anterioridad los momentos u eventos ante los cuales las funcionalidades volátiles deben ser activadas y/o desactivadas. Mediante este marco de trabajo esto puede ser programado de manera sencilla, lográndose así la visualización de los resultados correspondientes en tiempo de ejecución.

Se ha observado que este marco de trabajo encaja y podría ser aplicado perfectamente a aplicaciones web de empresas dedicadas a la comercialización de productos y servicios. Este tipo de aplicaciones son muy dinámicas y suelen sufrir modificaciones debido a la llegada de fechas especiales en las cuales suelen lanzarse promociones y descuentos de todo tipo. La utilización del marco de trabajo propuesto podría resultar muy útil para este tipo de eventos, ya que facilitaría, mejoraría y automatizaría la incorporación (y posterior remoción) de estas funcionalidades volátiles.

Como ha sido demostrado, este enfoque no ha quedado sólo en una definición teórica y conceptual, sino que ha podido ser llevado a la práctica satisfactoriamente. Se han implementado prototipos para los lenguajes de programación Java y Smalltalk. Así como esto resultó posible, lo mismo podría lograrse con cualquier otro lenguaje que sea capaz de cubrir los requerimientos que han sido definidos para este marco de trabajo.

Por último, resulta importante mencionar que tras realizar un experimento para medir la aceptación y el entendimiento del enfoque presentado, los resultados obtenidos han sido altamente positivos. La mayoría de los participantes se mostró interesada y tuvo buenas críticas para con el enfoque presentado, resaltando las mejoras y la conveniencia de su utilización.

12 Referencias

[1] Modeling, deploying, and controlling volatile functionalities in web applications. Matías Urbieta, Gustavo Rossi, Damiano Distante, Jeronimo Guinzburg. International Journal of Software Engineering and Knowledge Engineering 22(1): 129-155 (2012).

[2] Managing volatile requirements in web applications. Matias Urbieta, Gustavo Rossi, Damiano Distante, Wieland Schwinger: WSE 2013: 77-82.

[3] Oblivious Integration of Volatile Functionality in Web Application Interfaces. Jeronimo Ginzburg, Damiano Distante, Gustavo Rossi, Matias Urbieta: J. Web Eng. 8(1): 25-47 (2009).

[4] Transparent Interface Composition in Web Applications. Jeronimo Ginzburg, Gustavo Rossi, Matias Urbieta, Damiano Distante:ICWE 2007: 152-166.

[5] Spring Framework, http://docs.spring.io/spring/docs/4.2.0.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle/

[6] AOP, http://docs.spring.io/spring/docs/2.5.4/reference/aop.html

[7] Saxon XSLT, http://www.saxonica.com/documentation

[8] Maven, http://maven.apache.org/guides

[9] Drools, http://drools.jboss.org/documentation

[10] Modeling, deploying, and controlling volatile functionalities in web applications (pp. 20-22). Matías Urbieta, Gustavo Rossi, Damiano Distante, Jeronimo Guinzburg. International Journal of Software Engineering and Knowledge Engineering 22(1): 129-155 (2012).

[11] Web application. Daniel Nations. http://webtrends.about.com/od/webapplications/a/web_application.htm

[12] Sample de Spring Web Flow: booking-faces, https://github.com/spring-projects/spring-webflow-samples/tree/master/booking-faces

[13] Sample de Grails: petclinic, https://github.com/grails-samples/grails-petclinic

[14] Web Application Models are more than Conceptual Models. Gustavo Rossi, Daniel Schwabe , Fernando Lyardet: WWWCM99.

[15] Aplicando Estrategias de Mapeo OOHDM–WCML para el desarrollo de Aplicaciones Web. Germán Terrazas, Gustavo Rossi, Fernando Lyardet, Christian Segor.

[16] Guvnor-Drools, http://guvnor.jboss.org

[17] Seaside, http://www.seaside.st/documentation

[18] PHANtom (PHaro Aspect language), http://pleiad.cl/research/software/phantom

[19] Dynamic Aspects, http://scg.unibe.ch/archive/masters/Strau08a.pdf

[20] Design Patterns: Elements of Reusable Object-Oriented Software; Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; Addison-Wesley Professional Computing Series.

[21] Hot-Spot-Driven Framework Development. Wolfgang Pree. http://unisalzburg.at/fileadmin/multimedia/SRC/docs/publications/J016.pdf

[22] Software Development Life Cycle (SDLC) phases. http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/
[23] Software Maintenance As Part of the Software Life Cycle. http://hepguru.com/maintenance/Final_121603_v6.pdf
[24] Aspect-Oriented Programming. Niklas Påhlsson. http://oberon2005.oberoncore.ru/paper/np2002.pdf
[25] JavaScript Regular Expressions. W3Schools. http://www.w3schools.com/js/js_regexp.asp
[26] An Overview of Feature-Oriented Software Development. Sven Apel, Christian Kastner. http://www.cs.cmu.edu/~ckaestne/pdf/JOT09_OverviewFOSD.pdf
[27] XSLT Tutorial. W3CSchools. http://www.w3schools.com/xsl/
[28] XSLT: Transformaciones XSL. http://www.mclibre.org/consultar/xml/lecciones/xml_xslt.html
[29] Data Mapper, Patterns of Enterprise Application Architecture. Martin Fowler. http://martinfowler.com/eaaCatalog/dataMapper.html
[30] RulesEngine, Domain Specific Languages. Martin Fowler. http://martinfowler.com/bliki/RulesEngine.html
[31] SqueakSource Scheduler, http://www.squeaksource.com/Scheduler.html
[32] Pharo, http://www.seaside.st/download/pharo
[33] Glorp, http://www.eli.sdsu.edu/SmalltalkDocs/GlorpTutorial.pdf

13 Apéndice A: Instalación de prototipos

Para la instalación de los prototipos desarrollados de VF Framework se dispone de dos opciones:

- Instalación de imagen para máquina virtual
- Instalación manual

13.1 Instalación de imagen para máquina virtual

13.1.1 Herramientas necesarias

Para llevar a cabo la ejecución de los prototipos requeriremos el software para virtualización de arquitecturas VirtualBox. El mismo será utilizado para correr la imagen de una máquina virtual que posee todo lo necesario para la realización de pruebas.

Puede descargarlo desde: https://www.virtualbox.org/wiki/Downloads

13.1.2 Instalación

La imagen de la máquina virtual se encuentra comprimida en el CD entregado y se llama "LubuntuVF.vdi.zip". Descomprima dicho archivo, copie la imagen contenida a su computadora y siga los siguientes pasos para instalar la máquina virtual:

- Ejecutar VirtualBox.
- Presionar el botón "Nueva" del menú principal
- Ingresar:
  - Nombre: Lubuntu VF
  - Tipo: Linux
- Versión Ubuntu (32 bits)
- Presionar el botón "Next".

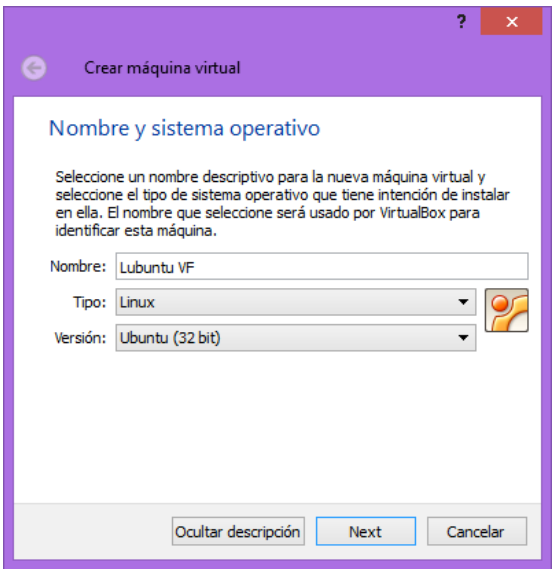


Ilustración 87: Instalación de imagen para máquina virtual 1

- Seleccionar como mínimo 1.5GB (1536 MB) como tamaño de memoria RAM.
- Presionar el botón “Next”.

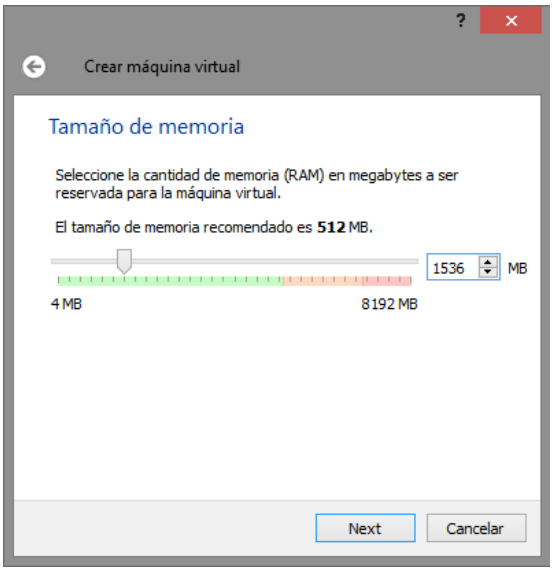
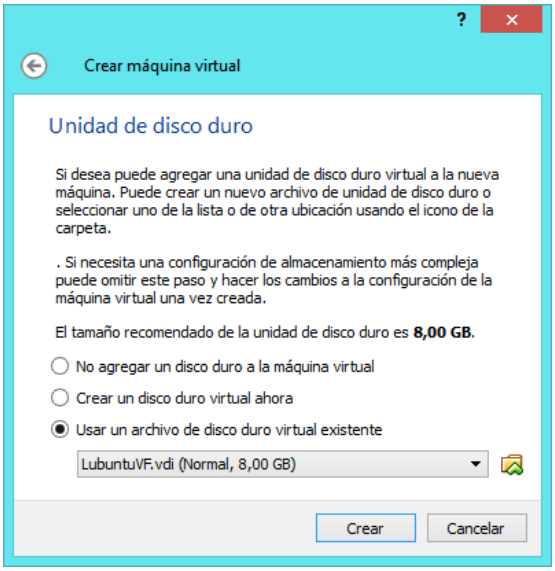


Ilustración 88: Instalación de imagen para máquina virtual 2

- Seleccionar la opción “Usar un archivo de disco duro virtual existente”.
- Seleccionar la imagen “LubuntuVF.vdi”.



- Seleccionar la máquina virtual creada y presionar el botón “Configuración”.
- Acceder a “Sistema” > “Procesador” y tildar la opción “Habilitar PAE/NX”.

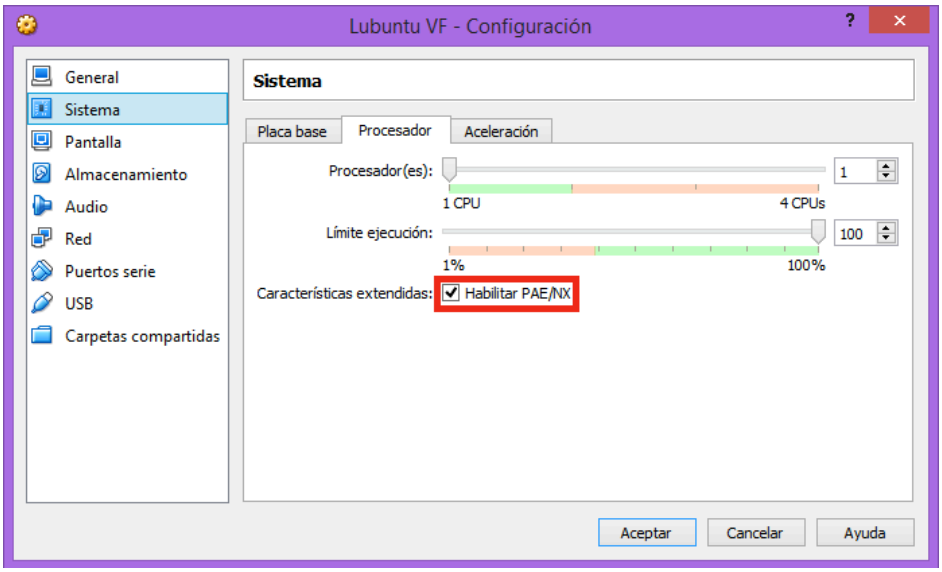


Ilustración 89: Instalación de imagen para máquina virtual 3

- Acceder a “Pantalla” > “Video” y tildar la opción “Habilitar aceleración 3D”.

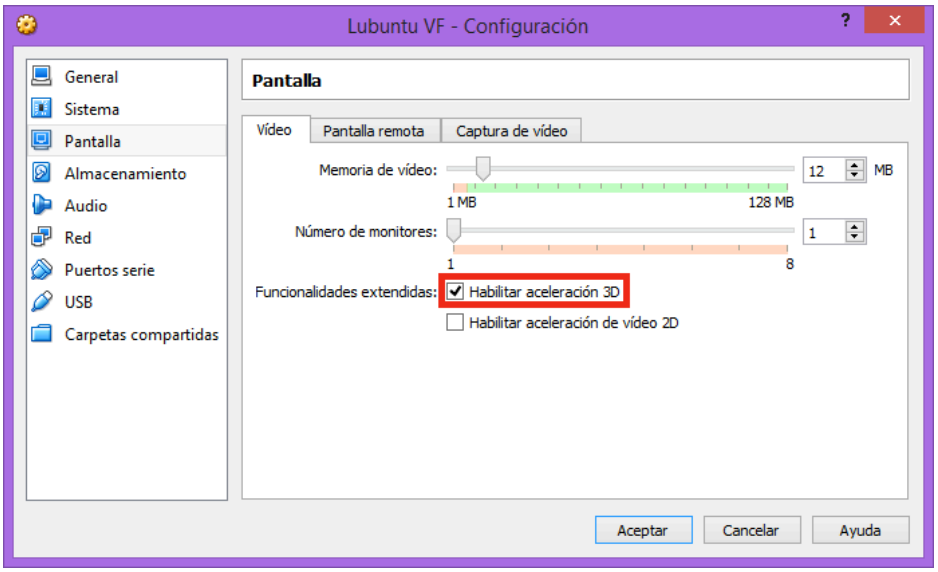


Ilustración 90: Instalación de imagen para máquina virtual 4

- Seleccionar la máquina virtual creada y presionar el botón “Iniciar”.

El usuario de la maquina virtual es “adminuser” y la password también es “adminuser”.

El prototipo en Java puede ser accedido a partir del entorno de desarrollo integrado Spring Tool Suite. Para acceder a dicho entorno debe hacer doble click sobre el archivo ejecutable “STS”, que se encuentra ubicado en el directorio Desktop/sts-bundle/sts-3.6.3.RELEASE. Dado que este prototipo utiliza una base de datos, también se ha dispuesto en el escritorio un acceso directo al editor visual de base de datos MySql Workbench.

El prototipo en Smalltalk puede ser accedido a partir del entorno de desarrollo integrado Pharo. Para acceder a dicho entorno debe hacer doble click sobre el archivo ejecutable “pharo”, que se encuentra ubicado en el directorio Desktop/Seaside3.1-OneClick.app/Contents/Linux.

13.2 Instalación manual

13.2.1 Instalación manual del prototipo en Java

13.2.1.1 Herramientas necesarias

Para llevar a cabo la instalación del prototipo en Java se requiere:

- JDK 7 (http://www.oracle.com/technetwork/java/javase/downloads)
- IDE Java EE con plugin de Maven tal como Spring Tool Suite (http://spring.io/tools/sts/all)
- MySQLServer (http://dev.mysql.com/downloads/mysql)
- Workbench (http://dev.mysql.com/downloads/workbench)

13.2.1.2 Instalación

Luego de instalar todas las herramientas requeridas, puede encontrar el prototipo implementado de la aplicación booking-faces con funcionalidades volátiles en el CD entregado con el nombre “booking-faces-vf.zip”.

Este archivo comprimido ZIP contiene:

- booking-faces: Código fuente del proyecto original.
- vf: Código fuente del proyecto que decora al proyecto original, añadiéndole las funcionalidades volátiles.
- init.sql: Archivo con la creación e inicialización del esquema en la base de datos.

Para probar el prototipo implementado debe acceder a Spring Tool Suite e importar los dos proyectos. Para ello debe:

- Descomprimir el archivo ZIP descargado. El código fuente de los dos proyectos contenidos debe ser ubicado en el directorio que será utilizado como workspace.
- Elegir el directorio de trabajo correspondiente al iniciar el Spring Tool Suite.
- Importar los proyectos desde Spring Tool Suite.

Para crear e inicializar el esquema de la base de datos que precisa la aplicación booking-faces debe acceder al Workbench e importar el archivo init.sql que se encuentra en el archivo ZIP descargado. La base de datos debe poder ser accedida a través del usuario "root" y sin password.

13.2.2 Instalación manual del prototipo en Smalltalk

13.2.2.1 Herramientas necesarias

Para llevar a cabo la instalación del prototipo en Smalltalk se requiere:

- Ambiente de desarrollo Pharo 2.0 con Seaside 3.1.

Para descargarlo e instalarlo de manera simple y rápida se recomienda la imagen preparada de Pharo "Seaside One-Click Experience 3.1". Esta imagen de Pharo está lista para ser utilizada y ya contiene Seaside incorporado. La misma puede ser descargada de la siguiente URL: http://www.seaside.st/download/pharo

13.2.2.2 Instalación

- El código fuente correspondiente al prototipo de la aplicación SushiNet Store y al VFFramework puede ser obtenido en el CD entregado con el nombre "sushistore-vf.zip".
- Este archivo comprimido ZIP contiene los siguientes paquetes:
- Scheduler-SeanDeNigris.22.mcz: Código fuente de Scheduler de SqueakSource [19].
- Store-Model.st: Código fuente del modelo de la aplicación original.
- Store-Seaside.st: Código fuente del controlador y de la vista de la aplicación original (implementados con Seaside).
- VFFramework.st: Código fuente del Framework provisto para permitir la implementación y manipulación de funcionalidades volátiles.
- Store-VFFramework.st: Código fuente de la implementación y configuración de las funcionalidades volátiles del prototipo, a partir de la extensión de los hotspots del VFFramework.

Para probar el prototipo implementado debe acceder al archivo ejecutable de Pharo correspondiente a su plataforma.

Todos los pasos necesarios para la instalación del prototipo serán detallados. Es muy importante que respete el orden en el que efectúe los mismos. De lo contrario, los paquetes podrían ser cargados de manera incorrecta.

Instalación de PHANtom

Para descargar PHANtom (PHaro Aspect laNguage), lo primero que deberá hacer será descargar su configuración. Para ello, deberá abrir un Workspace y ejecutar lo siguiente:

```
Gofer new
  squeaksource: 'Phantom';
  package: 'ConfigurationOfPhantom';
  load.
```

Luego, concluirá con la instalación ejecutando en el Workspace lo siguiente:

```
(ConfigurationOfPhantom project version: ‘1.2.3’) load: 'ALL'.
```

Instalación de prototipo y de VF Framework:

Estos archivos deberán ser importados desde el ambiente Pharo en el siguiente orden:

- Scheduler-SeanDeNigris.22.mcz
- Store-Model.st
- Store-Seaside.st
- VFFramework.st
- Store-VFFramework.st

Configuración de identificación de usuario autor

El proceso mediante el cual el VF Framework activa y desactiva funcionalidades volátiles consta de numerosas compilaciones. Por esta razón, es muy recomendable configurar de manera definitiva la identificación del usuario autor de Pharo. De lo contrario, la misma será solicitada en cada compilación requerida. Para configurar el usuario, deberá ejecutar el siguiente código en el Workspace:

```
Author fullName:'Darian Frajberg'.
```

14 Apéndice B: Configuración y prueba de procesador XSLT

El prototipo desarrollado para Java utiliza transformaciones XSL para la modificación de archivos de vistas y de archivos de configuración. El proceso de transformación de estos documentos (que deben ser de tipo XML) es realizado en compilación al efectuar el weaving mediante un plugin de Maven. Sin embargo, a la hora de implementar un nuevo archivo XSL resulta conveniente poder probar si se ha obtenido el resultado esperado luego de ejecutar la transformación. Para poder realizar esta prueba puede descargar el procesador Saxon XSLT 2.0 del siguiente enlace: https://ffmap.googlecode.com/files/saxon9he.jar

Luego de descargar el procesador, debe añadirlo como procesador XSLT de Java en el IDE que está utilizando.

En caso de utilizar Eclipse, esto se realiza accediendo a Window>Preferences>XML>XSL>Java Processors. Allí presione el botón “Add”. En la ventana emergente ingrese como nombre para el transformador “saxon 2.0”, elija como tipo de procesador “Saxon XSLT 2.0” y por último presione el botón “Add External JARs” y seleccione el archivo que ha descargado (saxon9he.jar). Luego presione el botón “Ok” y seleccione como procesador por defecto al procesador recién agregado.

Ya ha descargado y configurado el procesador XSLT. Ahora sólo resta correr las transformaciones sobre los archivos originales y comprobar que los resultados obtenidos sean correctos.

En caso de utilizar Eclipse, debe hacer click derecho sobre el archivo original sobre el cual se desea realizar la transformación. Se abrirá un menú dentro del cual debe seleccionar Run As>Run Configurations. Haga doble click sobre XSL. El XML Input File será el que ha sido elegido, mientras que el archivo de transformación XSL debe ser agregado clickeando en “Add Files”. Ahora presionando el botón “Run” finalmente podrá observar el nuevo archivo generado como resultado. Es conveniente completar el nombre de la transformación para identificarla más fácilmente cuando desee ejecutarla nuevamente en el futuro.