Volatile Functionality in Action: Methods, Techniques and Assessment

Darian Frajberg[1,2(✉)], Matías Urbieta[2,3], Gustavo Rossi[2,3], and Wieland Schwinger[4]

[1] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy
darian.frajberg@polimi.it
[2] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{murbieta, gustavo}@lifia.info.unlp.edu.ar
[3] Conicet, Buenos Aires, Argentina
[4] Department of Cooperative Information Systems, Johannes Kepler University Linz, Linz, Austria
wieland.schwinger@jku.ac.at

Abstract. One of the main features of most Web applications today is their great dynamism. They are undoubtedly characterized by a continuous evolution. After implementing and performing the first deployment of a Web application, some new requirements are bound to arise, which involve the need to incorporate new functionalities, generally unknown during the design stage. This type of functionalities, which arise as a response to unexpected requirements of the business layer, have the peculiarity that they eventually need to be removed due to the expiration of their commercial value. The continuous incorporation and removal of these functionalities, which we will call "volatile functionalities", usually has a negative impact on some important aspects of the Web application. Volatile Functionality meta-framework is a conceptual framework that permits to support the lifespan of volatile functionalities in Web applications. We have developed diverse techniques enabling full support of volatile functionalities for enterprise application. Moreover, we have performed an evaluation for assessing developers' experience and solutions' performance.

Keywords: Volatile functionality · Evolutionary architecture · Web application · Approach · VF framework · Event scheduling

1 Introduction

One of the main features of most Web applications today is their great dynamism. They are undoubtedly characterized by a continuous evolution. After implementing and performing the first deployment of a Web application, some new requirements are bound to arise, which involve the need to incorporate new functionalities, generally unknown during the design stage. Sometimes said functionalities are tested during a specific period and are then discarded as consequence of not having resulted to be useful enough for the users. Some other times, they appear as a response to determined events and/or conditions. Finally, it is quite common for certain functionalities to be

A. Bozzon et al. (Eds.): ICWE 2016, LNCS 9671, pp. 59–76, 2016.
DOI: 10.1007/978-3-319-38791-8_4

periodically activated at a specific moment of the year, and then be deactivated. That is to say, they are repeatedly introduced and removed from an application. This type of functionalities, which arise as a response to unexpected requirements of the business layer, have the peculiarity that they eventually need to be removed due to the expiration of their commercial value.

The continuous incorporation and removal of these functionalities, which we will call "volatile functionalities" (VFs), usually has a negative impact on some important aspects of the Web application. This occurs because usually the task is carried out manually and is quite prone to errors. Therefore, both the quality of the application code as well as its performance are harmed, requiring extra work for the developers during the maintenance stage, and probably a project cost overrun. It is well worth mentioning that VFs can be extremely complex and involve changes in all the application layers (business layer, navigation layer and presentation layer).

A clear example can be seen in the Web applications of companies engaged in online commercialization of products and services. The Black Friday launched by Amazon.com case is a marketing term employed to refer to the day following Thanksgiving in the United States, but has been adopted worldwide. Big companies created this event in order to encourage online shopping by applying great discounts and the same marketing strategy has been adopted for other promotion such as Cyber Monday case. Figure 1 shows the "Black Friday" promotions. On top of the image you can see several Black Friday banners notifying the availability of promotions. In the middle, there is a carousel showing products that have big discounts. Finally, there is another banner with the remaining time for the promotion, and one of the advertised products with its corresponding discount.

These components allow the navigation towards specific products on sale. Amazon's customization for "Black Friday" not only required the modification of Web pages through the introduction of images and links. Some other things also needed to be considered, such as the addition of new specific business rules in order to make the

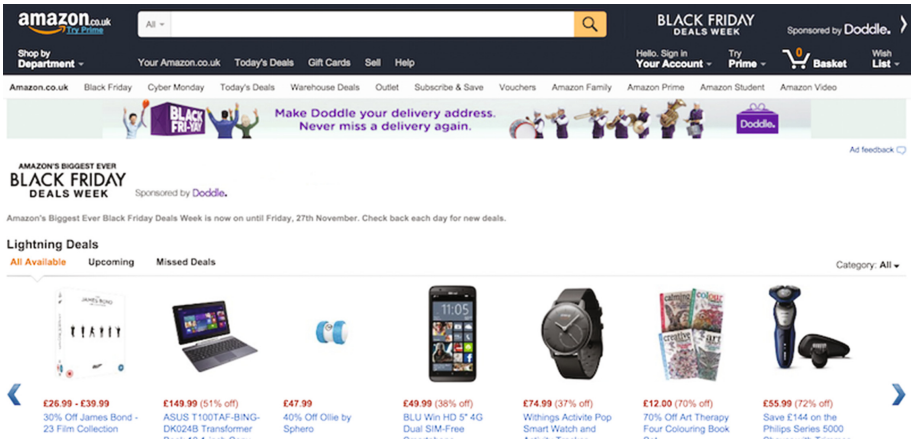

Fig. 1. Black Friday in Amazon.co.uk.

advertised discounts, the record of the sales with their corresponding promotions, and the specific definition of discounts for the different products involved, among others.

Considering the unforeseen and unexpected characteristic of volatile functionality (VF), its design and implementation introduce challenges to software engineers, as they were not considered during application design. Nowadays there are several approaches and tools for tackling challenges introduced by VFs, but they face the problem only partially. Aspect Oriented Programming (AOP) [4] allows the introduction of new variables and methods to classes, but does not consider its weaving and unweaving based on business events. On the other hand, Web pages can be augmented using document transformation technologies (XSLT [30]) but those available do not replace AOP for classes modification. In this work, we complement our approach for designing VF [17] with a meta-framework for instrumenting a new functionality modeled with the approach. This meta-framework defines the requirements that a platform must meet for supporting unexpected and unforeseen requirements with a seamless solution.

To cope with the above mentioned problems, we present as novel contribution:

- A set of basic requirements an application that supports VF lifecycle must meet. All the minimum fundamental concepts which software solutions are required to satisfy in order to make use of such framework.
- A discussion about how those requirements were introduced into two applications, one of them based on a static type binding language and the other one on a dynamic type binding language. This meta-framework proposes the decoupling of VFs from the original application, thus maintaining the integrity and independence of the latter, where the type of binding determines the implementation strategy.
- Finally, an experiment has been conducted in order to evaluate and obtain feedback from impartial developers regarding the presented framework.

The rest of the paper is structured as follows: in Sect. 2, we discuss some related work; in Sect. 3, we introduce the approach; in Sect. 4, we present the VF Framework and an evaluation using Java and Smalltalk technologies; and in Sect. 5 we describe an evaluation for assessing developers' experience and solutions' performance; finally in Sect. 6, we present the conclusions of our work.

2 Related Work

Systems evolution has been largely studied in order to avoid common mistakes, predict costs of maintenance and reduce changes' impact. In [7], a study of variability is presented as the ability to change software artifacts for specific context. Small and Medium Enterprises were surveyed resulting as outcome an enumeration of challenges such as documentation, change traceability, change extent, among others when facing changes in a system. Unfortunately, change impact analysis was not deepened in products' source code and quality.

An outstanding trend to deal with transparent evolution is the use of Aspect-oriented technologies as mentioned some paragraphs above. In the specific area of Web applications, an interesting approach can be read in [1]. In this work the authors propose to tackle evolution using Aspect-oriented Design Patterns. The approach is

sound and powerful as the authors identify a set of possible types of changes in a Web application and associate an Aspect Pattern to solve each specific situation. However, it is geared towards implementation more than design and therefore it loses part of its power; nevertheless, as discussed throughout this paper, the underlying ideas behind Aspect orientation are key concepts to use with VF.

Another related work in which we found similar problems and ideas to ours is Web application adaptation. A Web modeling framework with orthogonal facilities for extending functionality in a seamless way is AMACONT [11]. This framework provides means for addressing adaptation in Web applications by implementing AOP concepts. Using aspect-oriented adaptation and semantics-based adaptation for different adaptation granularity, it allows specifying changes in a component-based model. The work provides a useful framework but lacks of means for specifying presentation aspects as well as describing aspects lifecycle.

The traceability of changes in the lifecycle of evolution/maintenance requirements has been combined with Impact Analysis (IA) in [6]. It provides an integrated approach that gives traceability support from the change request, going through model artifacts, source code changes and execution behavior. This approach can be used for tracing any Volatile Requirement, but its removal is still compromised and should be studied.

Model Driven Engineering aims at providing methods and tools for designing Web applications, where developers abstract from source code aspects and focus on functional requirements instead. In [9], authors studied how maintenance tasks, in both OOHRIA approach and code centric approach, were perceived by practitioners. The experiment was performed by 26 students and their perception was captured by questionnaires. The results showed that OOH4RIA improved efficiency and effectiveness of maintenance tasks over the same tasks when using .NET technology. However, the abovementioned evaluations were focused on subjective metrics gathered from students' perception and they did not cover source code's metrics in order to assess internal quality such as object-oriented metrics [2] for code centric solution and model metrics. Additionally, most of the empirical research works focused on the development from the scratch study case whereas we study maintenance challenges.

Technical debt has been studied largely in industry projects [3] and how to manage it [14] as it highly relates with an application's budget. In this field we can find plenty of tools that enable to measure technical debt from source code [21, 24, 25, 29], but also taking into account other points of view such as architecture-sourced debt [12] on components coupling [10] or database schemas' technical debt [18]. Technical debt must be analyzed using different approaches in order to gain as much information as possible. In [20], Hadoop an open source No-sql database was processed using code smells, automatic static analysis issues, grime buildup, and modularity violations taking into account several application versions. The outcome remarks that techniques are loosely coupled and therefore a Technical debt analysis should combine several techniques.

Feature Oriented Programming has been an approach for facing those developments that compose functionality mainly focused on Software Families. In [15], authors introduce the relevant factors in variability realization of Software product families. As part of the work, they proposed taxonomy of variability introducing different realization

techniques for software changes. The presented techniques were mapped to surveyed and/or studied software companies that adapt their software to changes.

Maintenance effort was studied in [8], where ten PhD and Master students were asked to maintain a desktop application. The work studied the processes and tools used by students for solving the problem in depth. The outcome was a characterization of problems, relevant code and APIs, and testing processes.

Though transparent improvement of conceptual and navigational models has been treated in the literature, we are not aware of any approach supporting oblivious [4] composition of interface design models in such a way that different concerns keep orthogonal. In the XML field, the AspectXML project [22] has ported some concepts of aspect-orientation to XML technology by allowing the specification of point-cuts and advices similarly to Aspect Java. The project is still in a research stage.

3 Approach

In most mature Web design approaches, such as UWE, WebML, UWA, Hera, OOWS or OOHDM (see [13] for description and examples of each approach), a Web application is designed with an iterative process comprising at least conceptual and navigational modeling. Most of these design methods produce an implementation-independent model that can be later mapped to different run-time platforms. For the sake of clarity we will concentrate on the conceptual, navigational and interface models, as they are rather similar in different design approaches.

As most of the problems discussed so far apply to all development approaches, we will first describe the philosophy underlying our technical solutions in such a way that it can be reused. In [17] we describe its usage with OOHDM design models and briefly discuss how each part of our approach could be adapted to other methods.

Our approach is based on the idea that even the simplest VF (e.g., a discount available for a period of time, as in Fig. 1) should be considered as a first-class functionality and, as such, designed accordingly. At the same time, their design and implementation have to be taken separately and as much as possible decoupled from that of core and stable functionalities.

Building on the above ideas, our approach can be summarized with the following design guidelines, which are shown schematically in Fig. 2:

- We decouple volatile from core functionalities by introducing a design layer for VFs (called Volatile Layer), which comprises a requirements model, a conceptual model, a navigational model, and an interface model.
- Volatile requirements are modeled using the same notation used to model core requirements (e.g., use cases, class diagrams, user interaction diagrams, etc.) and separately mapped onto the following models using the heuristics defined by the design approach. Notice that, as shown in Fig. 2, volatile requirements are not integrated into the core requirements model, therefore leaving their integration to further design activities.
- New behaviors, i.e. those which belong to the VF layer, are modeled as first class objects in the volatile conceptual model; they are considered as a combination of

Commands and Decorators [5] of the core classes. This strategy applies also to slight variants of business rules (such as adding a price discount to a product). In this case, the decoration is applied at the method level more than at the class level. Notice that this strategy can be applied to any object-oriented method, i.e., any method using a UML-like specification approach.

- We use inversion of control to achieve obliviousness; i.e., instead of making core conceptual classes aware of their new features, we invert the knowledge relationship. New classes know the base classes on top of which they are built. Core classes, therefore, have no knowledge of the additions. This also stands for aspect-oriented approaches.
- Nodes and links belonging to the volatile navigational model may or may not have links to the core navigational model. The core navigational model is also oblivious to the volatile navigational classes, i.e., there are no links or other references from the core to the volatile layer. This principle can be applied in any Web design approach.
- We use a separate integration specification to specify the connection between core and volatile nodes. As we show later in the paper, the integration is achieved at run-time. In other model-driven approaches, the integration can be performed during model transformation by implementing the corresponding transformations.
- We design (and implement) the interfaces corresponding to each concern (core and volatile) separately; the interface design of the core classes (described in OOHDM using Abstract Data Views (ADV) [16]) are oblivious with respect to the interface of volatile concerns. As in the navigational layer this principle is independent of the design approach.
- Core and volatile interfaces (at the ADV and implementation layers) are woven by executing an integration specification, which is realized using XSL transformations. Again, the idea of model weaving is generic and therefore the same result can be obtained using another technical solution.

In the next section we introduce methods and approaches for supporting VF in Enterprise Web applications.

4 VF Framework

In order to implement a system supporting VF, a set of non-functional requirements must be met. These requirements, when satisfied by the underlying technology of either a new or legacy application, permit to configure VF abstracting from activation/deactivation rules, seamless entities enrichments and Web page augmentations. We have already discussed how to manage VF in MDWE in [17].

We present this sort of checklist in Table 1, which helps architects to introduce new unforeseen and unexpected requirements on in-production systems. The approach used in this work aims at adopting new requirements, even though the involved architecture did not consider them while it was designed. Such approach is available for Object-Oriented systems that implement the MVC pattern.

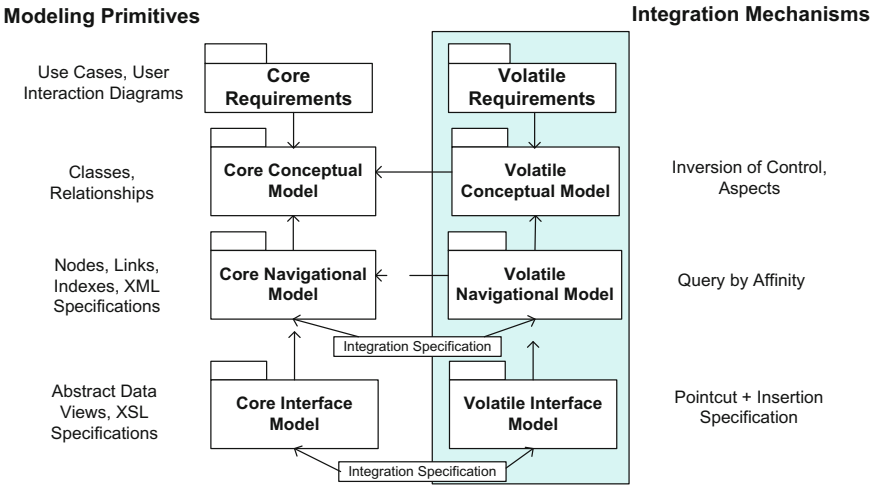

Fig. 2. A schematic representation of our approach

Two prototypes[1] were implemented in order to test the presented framework. One of them was developed by using Java and Java Enterprise Edition (JEE) frameworks, and the other one through Smalltalk. Their implementation pursued not only a concrete and practical demonstration of VF Framework, but also the analysis and testing of diverse strategies in order to achieve the incorporation and manipulation of VFs in Web applications. The selection of these programming languages was based on the fact that they have important differences with each other. While Java is a static typed language, Smalltalk is a dynamic typed and pure Object-Oriented one.

4.1 Weaving Approach

Architecture plays a main role, as it must support the VF's approach artifacts. Any new unexpected and unforeseen requirement must be thought independently from the core application and specify the weaving mechanisms for its compromised application layer. That is, the core application's code must be kept untouched and any augmentation must be automated. Depending on the underlying technology there are two possible weaving approaches, which are presented next:

Dynamic: Dynamic weaving approach is a strategy available for powerful and dynamically Web applications technologies, which are capable of executing the weaving between the core application and the VFs on the fly. It permits introducing changes at run-time without need of recompiling.

The Smalltalk implementation uses the dynamic weaving approach for the incorporation of VFs. The code corresponding to a VF is located in a package of the running Virtual Machine image and the code weaving is indeed performed dynamically by modifying the application model at run-time.

[1] Source code, demo video and used questionnaires are available at https://goo.gl/J7C2WV.

Static: Static weaving approach is a strategy for those Web applications technologies that do not permit the augmentation of their artifacts (i.e. classes and UI) at run-time. When using this approach, the weaving between the core application and the VFs is executed at compilation-time. Therefore, all the introduced VFs should be configured before the deployment. Otherwise, the application will need to be restarted. Moreover, as the incorporated modifications are introduced statically, they should have the capacity to evaluate the state of the corresponding volatile functionalities and behave in accordance to them.

The JEE implementation uses the static weaving approach. Java Enterprise Edition uses several languages for implementing Web from Java, which is a strong typed and static language that requires type checking and a compilation process before running the code, to JSP, which combines HTML, Javascript and scriptlet content for rendering the user interface. This constraint requires a solution in which changes must be compiled and packaged, as it is done with the core application. Next, they also require to be weaved at compilation-time by a build engine such as Maven [26]. For such task we combine full-fledged technologies, each solving a specific technology weaving challenge. For example, the VFs are added mainly by defining a set of XSL transformations and aspects, which are all applied by Maven on the compiled original application. Thus, the VFs are not intrusive with the original application and can be added within the compilation cycle. The volatile code is packaged in a War file with the new classes, Web pages and artifacts.

Table 1. Features required for supporting VF approach

Feature	Strategies	Applicability conditions	Implementation examples
Weaving approach	Dynamic	Dynamic type binding	Reflection
	Static	Static type binding	Maven
Business & Navigation model enhancement	AOP	Methods interception	AspectJ, PHANtom
	Regex modifications	Interpreted code execution	Pharo native support
Presentation model enhancement	Documents transformation	Document-based UI	XSLT
	AOP	Methods interception	AspectJ, PHANtom
	Regex modifications	Interpreted code execution	Pharo native support
Lifespan management	Rules-engine based	Mature enterprise solutions	Drools
	Scheduler and AOP based	Restricted development kit	Quartz

Conclusions: The approach implements an architecture that keeps core application oblivious from new changes, which must be integrated and managed as corresponds. The adoption of any of the weaving approaches is a decisive factor, which will have a direct impact regarding the applicability of the strategies to cover the other framework's basic features. Undoubtedly, the dynamic weaving approach provides a more flexible and simple solution in order to accomplish the goal of supporting VFs.

4.2 Business and Navigation Models Enhancement

The modification of the business layer includes matters such as the injection and Object-Relational Mapping (ORM) of variables and relations, and the injection and modification of methods. On the other hand, the modification of the navigation layers comprises the injection of dependencies, and also the injection and modification of methods. The techniques to perform these changes are presented next:

AOP: Aspect Oriented-Programming [4] is a paradigm that aims at modularizing crosscutting concerns. This technology can be used for the enhancement of business and navigation models for those cases in which their implementations are defined by classes. AOP provides the capability to modify classes' structures by adding new variables and methods and also their behaviors by intercepting and modifying methods with associated pointcuts and advices. However, there is an important difference regarding how AOP is used depending on the programming languages' type systems. For dynamically typed languages, aspects can be enabled and disabled at run-time as needed, while for statically typed languages, this is not possible as aspects have to be incorporated during compilation-time. Therefore, the modification of methods for static languages must be controlled by wrapping the advices' code within conditional sentences based on the current state of certain VF. Thus, the involved classes will behave as appropriate. It is not mandatory to perform the same treatment for injected methods due to the fact that when their corresponding VFs are disabled they should not even be invoked.

Both developed implementations use aspects. The JEE implementation uses AspectJ framework and, as it is a statically typed programming language, it incorporates the modifications at compilation time. As for the ORM of injected variables, the Java prototype has covered it perfectly through Hibernate annotations. The Smalltalk implementation uses PHANtom [27] framework and the ORM of variables has not been taken into account for it. This is due to the fact that it does not employ a real database, but is managed as from instances created in memory. Nevertheless, it has been analyzed and should be covered perfectly by using the Glorp [19] ORM technology. Code 1 and Code 2 present examples of business model enhancement for JEE and Smalltalk implementations, in which a variable that holds a discount, as well as a getter and a setter for it, are injected into a class. Moreover, they both modified the computation of the final price by intercepting it and applying the corresponding discount.

Regex Modifications: Regex modifications aim at using regular expressions for the identification of fragments of source code corresponding to certain classes' methods.

Afterward, the defined modifications for them can be effected in run-time. Logically, regex modifications can only be implemented for interpreted or dynamic type binding programming languages such as Smalltalk and PHP, which must be capable of modifying the classes' model with dynamic code generation.

The Smalltalk implementation also uses regex modifications to alter the original flow of existing methods. These modifications are executed within the framework's provided predefined methods used to fill the VFs' hotspots.

Conclusions: AOP demonstrated to be capable enough to support the features corresponding to the business and navigation layers for languages in which their implementations are defined by classes. Nonetheless, languages with interpreted code execution have some important advantages towards those that do not, provided that they have two options in order to modify the flow of existing methods. The first one is the interception with associated pointcuts and advices; and the second one is the use of regex modifications. Although interception through aspects can be sufficient in some cases, it can be inconvenient when multiple VFs involve modifications of coincidental code sections. Advices can be defined to be executed before, after or at the time of accessing a determined pointcut. Therefore, the correct execution of a method intercepted by multiple VFs is limited by the degree of modularization that it possesses. On the other hand, regex modifications can deal much better with this issue since with them it is able to identify and modify fragments of source code within methods.

```
public aspect DiscountVFAspect {
@NotNull
private int   Product.discount = 0;

public int Product.getDiscount()
    {return this.discount;}

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

pointcut getFinalPricePointcut(Product p) : exe-
cution(* Product.getFinalPrice()) && target(p);

int around(Product p):getFinalPricePointcut(p) {
    return proceed() * p.getDiscount(); }
}
```

Code 1. Aspect definition using AspectJ in JEE implementation.

```
AddVars
self addInstVar:'discount' toClass:'Product'.

addMethods
self addInstMethod:'discount
    ^discount ' toClass: Product.
self addInstMethod:'discount:aFloat
    discount:=aFloat ' toClass: Product.

addPointcuts
self add:(PhAdvice new pointcut:
(PhPointcut new receivers: Product;
selectors: 'finalPrice'; context: #(#receiver
#arguments #selector #proceed));
advice: [:context | context proceed * self
discount.]; type: #around).
```

Code 2. Ad-hoc framework in Smalltalk for supporting VF features.

4.3 Presentation Models

The modification of the presentation layer implies the alteration of the views that comprise the Web application interface. Once the changes to the model have been made, it is essential for said changes to be reflected in the interface to be consumed by

the final user. This task consists of effecting the corresponding transformations in each view and visualization, depending on the state of the VFs included in the application (activated or deactivated). The resulting application must be capable of showing or hiding the changes introduced at run-time. In order to achieve this aim, there are three possible strategies, which are the following ones:

Documents Transformation: The documents transformation implies the use of transformation pipeline, in which a document used as UI is augmented by a set of transformations. These transformations are an option for those languages that use documents such as XHTMLs for the views. It is important to consider the capacity of the used technologies to modify the processing pipeline of each request, as it will define if the transformations can be executed in run-time as necessary. For example, in order to introduce transformations in Servlet-based technology, we need to introduce Servlet-filters responsible for transforming documents. In that case, each time a VF is activated or deactivated, the views engaged must be modified in the act. Otherwise, changes should be incorporated during compilation-time. For that purpose, the modifications should include a conditional visualization of the modifications, which implies the introduction of conditional sentences (present throughout the whole execution of the application) responsible for visualizing or not the modifications belonging to the VFs. Logically, for this to be possible the views need to have the capacity to process information on the server side so as to make decisions based on it. Thus, the state of each VF is evaluated at the moment a modified view is invoked. If the VF is activated, all the changes introduced in the view will be presented. Otherwise, the original view will be shown.

The JEE implementation uses XSLT transformations of XHTML documents, which are executed in compilation-time through conditional visualizations. An example of this can be seen in Code 3, where a banner is displayed depending on the VF's state.

AOP: There are technologies such as GWT or Seaside that permit to specify UI programmatically using an API. As discussed previously, AOP perfectly fits in this scenario for enriching interface aspects by executing a piece of code (advice) before, after or both a given method is invoked.

The Smalltalk implementation uses aspects in order to incorporate new styles for the views.

Regex Modifications: As mentioned above, UI defined programmatically with an API can be modified by changing the source code, when coded with interpreted languages. The Smalltalk implementation uses regex modifications to introduce new elements to views and styles at run-time. In Code 4 there is an example of regex modification for the augmentation of Seaside user interface by adding a promotion banner.

Conclusions: The modification of the presentation layer in dynamic typed languages where the enhancement targets are classes turns out to be a much simpler, powerful and scalable solution than in those that use view documents. On one hand, aspects and regex modifications are easy to define (being regex modifications the most convenient option). On the other hand, documents transformations are much more complex and may become a bit cumbersome when they are implemented. Moreover, high complexity can also bring great difficulty regarding the modifications' scalability.

Nevertheless, it should also be taken into account that executing transformations at run-time might turn out to be slow and inefficient if the employed architecture does not count with the adequate processing speed.

```
<xsl:template match="node()[@id='banner']">
<xsl:text><![CDATA[<c:choose>
<c:when test="#{vf.enabled}">]]></xsl:text>
<h1 id="banner" Promotion 2x1</h1>
<xsl:text><![CDATA[<c:when>
<c:otherwise>]]></xsl:text>
<xsl:copy-of select ="."/>
<xsl:text><![CDATA[<c:otherwise>
<c:choose>]]></xsl:text></xsl:template>
```

Code 3. An example of JSP augmentation using XSLT.

```
addMethodsModifications
self addInstMethodModification: (
InstMethodModification  instMethod:
'renderContentOn:' forClass: Store
beforeRegex: 'html div id\: "body"'
insert: 'html table id: banner;
with: [html tableRow: [
html tableData: [
html div id: "title"; with:"Promotion 2x1" .] ] ].').
```

Code 4. An example of Seaside user interface augmentation using regex modification.

4.4 Lifespan Management

The lifespan management is one of the main features of the framework because it is responsible for VFs' activation and deactivation scheduling. It is required to count with a suitable technology for configuring three different types of events, which are: temporal events, business events and temporal business events. For example, a specific date, for enabling a promotion, and business based events, such as "Out of Stock", for disabling a promotion.

VF's orchestration manages the VF's activation/deactivation, but does not imply weaving as that is constrained to the underlying technologies introduced in Sects. 4.2 and 4.3. Nonetheless, if the platform allows it, then it may include the execution of hot deploys and weaving of VFs in real-time. This possibility is only available for certain technologies, which provide certain flexibility and avoid stopping and re-compiling the application in order to introduce a new VF.

For such purpose, the following strategies can be applied:

Rules-Engine Based: The activation and deactivation scheduling is perfectly achieved by using a rules engine. It makes it possible to configure all the types of events in a very simple and natural way.

The JEE implementation uses Drools [23] rules engine, which effectively enables to define any kind of rules for the activation and deactivation of VFs. In Code 5 an example for temporarily activating product discounts in such rules engine is presented. The rule will enable the VF on Feb 12th at 00:00 and will be online for two days. Said implementation does not allow hot-deploy of VFs, but needs to be compiled and deployed again in order to do add them. However, this constraint could be removed if VF is built on top of OSGI architecture.

Scheduler and AOP Based: The activation and deactivation events can also be covered by using a Scheduler and AOP. Temporal events can be configured with a

Scheduler such as Quartz [28], business events with aspects' interception and temporal business events with a combination of both Scheduler and aspects.

The Smalltalk implementation used this strategy provided that no good and well documented rules engine was found for this programming language. This approach required more work, but got similar final results to the Rules-engine based. Code 6 shows a Smalltalk-based sentence for enabling a VF, which will be enabled 10 min after its deployment. This implementation clearly covered the hot deploy without any complication due to its dynamism and flexibility.

Conclusions: We need to manage the activation of VF available in a given application as it distinguishes our approach from an ad-hoc one. The hot deploy of VFs is not a mandatory feature for the framework. Conversely, the activation and deactivation scheduling must definitely be covered. And we can assure that the usage of rules engines for scheduling activation and deactivation events seems to be the simplest and most suitable solution. However, if it is not possible to use one of them, the same results can be achieved with the Scheduler and AOP based approach.

```
rule "discountVF activation rule"
date-effective "12-Feb-2016 00:00"
date-expires "14-Feb-2016 00:01"
when eval(true);
then vfService.enable("discountVF");end
```

Code 5. Drools' rule for temporarily activating product discounts.

```
setUpScheduler
self scheduleEnableOnceAt:
(DateAndTime now + 10 minute).
```

Code 6. Smalltalk-based sentence for enabling VF.

4.5 Discussion

The implementation of prototypes by using different programming languages as Java and Smalltalk was very important for the development of this research. These prototypes allowed us not only to put the VF Framework concepts into practice, but also to consider different alternatives and strategies for their implementation.

Although both prototypes covered the most important VF Framework concepts satisfactorily (and might as well cover the remaining ones), a better result has been obtained with the Smalltalk prototype. Unlike Java, Smalltalk is a dynamically typed language, which makes it a very powerful alternative, and is very easy to handle. These characteristics have a great impact, as they enable a much more dynamic implementation and with more and better options to be carried out.

Furthermore, a white box framework has been developed for Smalltalk, which greatly facilitates the implementation process of VFs according to the VF Framework concepts. Simply from the extension of certain classes and the implementation of certain hotspots, it is possible to incorporate VFs in a decoupled manner and to schedule the activation and deactivation events for them. This task results more costly for Java, where the modifications and programming of events corresponding to the varied VFs are scattered without a clear separation from each other.

Besides, the use of AspectJ and Drools in Java obtained highly positive results and their incorporation was relatively simple. In the case of the Smalltalk implementation, it was necessary to search for new and more complex alternatives (combining different technologies) in order to achieve similar final results.

5 Evaluation

5.1 Experiment Planning

We have designed two experiments in order to study how different approaches perform at maintaining VF. Each experiment assesses the developers' perception of the development process as well as the software internal quality. Software components of code-based applications are affected by the introduction and later removal of VFs, we evaluate a set of representative quality metrics that covers business model and controller tiers, usually coded using programming languages such as Java, Python, C#, etc., and interface tier implemented with HTML, JavaScript and CSS.

By the introduction and later removal of such VF, Web applications pass through different development stages. The application was modified twice and, for each version, a snapshot was captured so as to evaluate its quality. We will call Original Version (OV) the starting stable version that resolves core business requirements. Next, we will have a version that introduces a set of VFs extending the application, which will be outlined later, called Volatile Version (VV); and finally we have a version that will be the result of removing such VFs, named Maintenance Version (MV), that presents the set of requirements served by OV. At first sight, MV should be alike OV and we will study any difference found focusing on those that are detrimental to the overall quality. The transition of Web applications' versions is shown in Fig. 3.

We have selected a hotel booking site, as a running example, which has been modified in order to introduce two VFs: Long weekend promotion (that includes the application of discounts and promotional interface enhancements) and Notification (that informs the authorities if a booking cost exceeds a predefined limit). We tackle

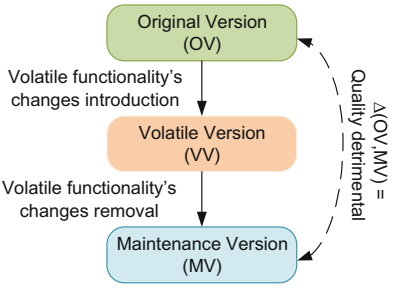

Fig. 3. Application's evolution scheme.

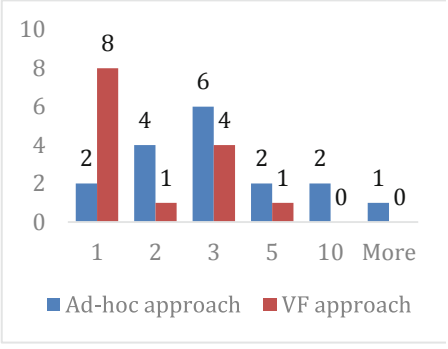

Fig. 4. Testing cycles per approach.

VF Implementation and maintenance by using both an ad-hoc application and VF approach. The first one requires manual management of VF lifecycle and code belonging to VF software artifacts, and the latter requires the use of the presented framework that fully supports VF's characteristics. For this application we have conceived VFs like those already discussed in [17].

Besides, we perform a tier-focused semi-automatic analysis of source code to assess internal quality. Variances between OV and MV will be analyzed and discussed.

5.2 Participants

The two experiments were performed by independent groups of developers. In the first one, we evaluated the maintenance by using an ad-hoc approach with 17 developers, and in the second one we used a VF approach with 14 developers. We aim at capturing VF maintenance challenges from the performers' point of view. They were students attending the last year at the National University of La Plata's bachelor degree (comparable to European Master Degree). Developers were mostly students with programming experience (about at least 4 years), who received the target applications in VV and were later asked to remove the VFs on their own, without any extra guide other than removing the deprecated functionality. After the developers removed the VFs and got the MV, we performed a quality evaluation of each version (OV, VV and MV) in order to detect whether changes were detrimental to the overall quality. Students were motivated and committed to the experiment, as it was part of their final examination earning education credit.

5.3 Research Questions

In order to evaluate VF approach performance against ad-hoc approach, we have defined research questions. Next, we present a summary of Research Questions, considered metrics and statistical methods.

(a) Is ad-hoc approach more prone to have application external quality issues?

Ad-hoc maintenance of VF showed in 76 % of the samples that perceivable changes (labels, library imports, message containers, etc.) were overlooked, leaving evidence of functionality associated to the VF. Using our VF Framework, the introduction and removal of VF did not modify core application and, as a result, there are not perceivable changes pending or quality issues.

(b) Is VF approach more productive than ad-hoc approach?

Yes, on average removing VF from applications with an ad-hoc (5.2 h) approach took almost five times more than removing with the VF approach (0,9 h). On the other hand, as confirmed before, using an ad-hoc way is prone to leave overlooked changes, being detrimental to internal software quality. By using a full-fledge framework for removing functionality, no line of code is left unattended. As we do not have enough samples, it is not possible to perform statistical analysis for significance.

(c) Is application's internal quality more negatively affected by Ad-hoc approach than VF approach?

Both business and controller layers were affected by quality issues. By running SonarQube tool, we detected several quality[2] issues from unused methods to commented code. This situation is detrimental to software quality as deprecated VF artifacts get tangled and scattered in the application making it more complex to understand by developer. Moreover, 94.12 % and 70 % of samples left unattended changes (variables, methods and metadata) at model and controller layers. As mentioned before, our framework permits to model new changes at business and controller layer without modifying code corresponding to the core application. When needed, it is removed by a tool ensuring that the core application's code is kept untouched.

(d) Is the developers' confidence affected by VF approach?

We measured the developers' confidence through survey questions that consider stability, complexity and maintenance. The results show that 86,7 % consider that using VF approach the application was equally or more stable than ad-hoc approach.

On the other hand, we also evaluated the required testing to complete the maintenance task. As presented in Fig. 4, the VF approach required less testing to check whether the VF has been completely removed. Most of the developers using VF approach claimed that one test allowed them to verify the correctness of the application, whereas with the ad-hoc approach developers needed 10 or more testing cycles.

5.4 Conclusions

Evidence shows that applications based on modern MVC and ORM frameworks may suffer from the discussed problems and this research sets the basis for further work considering other kinds of Web applications. The result shows that using a VF Framework avoided introducing errors when removing deprecated functionality, saved time and effort, and improved developer's experience skipping sloppy tasks. We were able to validate not only perceivable quality, but also internal one that compromised the technical debt. Additionally, developers involved found that development process was positively improved from experience and perception.

6 Conclusions

To sum up, we may highlight some interesting issues, which have arisen throughout the development of this research.

The proposed framework definitely improves the lifespan of VFs in Web applications. Moreover, it has been contributed with evidence that the latter fits and might perfectly be applied to enterprise Web applications, devoted to the marketing of products and services. This type of applications is very dynamic and often suffer modifications during certain special dates responding to business requirements, when

[2] Java rules http://dist.sonarsource.com/reports/coverage/rules_in_squid.html (Mar 15 2016).

all kinds of offers are launched. Using the proposed framework might be very useful for this type of events, as it would ease, improve and automate the management of VFs.

As shown, this approach has not limited itself to the theoretical and conceptual definition, but has been satisfactorily taken into practice by implementing two prototypes that exploit methods and techniques outlined by the framework, which overcomes VF implementation challenges. Just as this was possible, the same could be achieved with other languages capable of covering the defined requirements.

It is also important to mention that after having carried out an experiment to measure the acceptance and performance of the presented approach, the obtained results have been highly positive. The framework has indeed shown great advantages over the traditional approach for the management of VFs. The quality of the original application's source code is not degraded, no errors are introduced and time and costs are reduced. Furthermore, most of the participants showed interest and issued good reviews regarding the presented approach, highlighting the improvements and convenience of its utilization.

The future of the approach basically consists of deepening each of the concepts in order to obtain better results. So far, the two prototypes obtained a very positive outcome. However, these prototypes should continue to be improved, optimizing their performance, facilitating their implementation and undertaking some tasks left aside such as the testing of OSGI for the Java implementation and the study of different persistence frameworks. Furthermore, the development of new prototypes with new underlying technologies would be definitely meaningful for the research, as it could lead to the emergence of new strategies to support the defined framework's features.

Once solid and mature implementations have been consolidated, the next step might aim at the creation of frameworks that facilitate the support of VFs within Web applications. This point would be vital for the approach to be positioned and to start being used by the community in the industry.

Finally, the framework does not currently consider a graphic environment for the configuration of the activation and deactivation events. On the contrary, it uses mechanisms that can only be implemented by developers. The implementation of an intuitive environment would be important, as it would allow users without much IT knowledge to define their own activation and deactivation rules.

References

1. Bebjak, M., et al.: Evolution of web applications with aspect-oriented design patterns. In: Proceedings of ICWE, pp. 80–86 (2007)
2. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
3. Curtis, B., et al.: Estimating the size, cost, and types of technical debt. In: Proceedings of the 2012 3rd International Workshop on Managing Technical Debt, MTD 2012, pp. 49–53 (2012)
4. Filman, R.E.: Aspect-Oriented Software Development. Addison-Wesley, Boston (2005)
5. Gamma, E., et al.: Design patterns: elements of reusable object-oriented software (1995)

6. Gethers, M., et al.: An adaptive approach to impact analysis from change requests to source code. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 540–543 (2011)
7. Ihme, T., et al.: Challenges and industry practices for managing software variability in small and medium sized enterprises. Empir. Softw. Eng. 1–25 (2013)
8. Ko, A., et al.: An Exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans. Softw. Eng. **32**(12), 971–987 (2006)
9. Martínez, Y., et al.: Empirical study on the maintainability of Web applications: Model-driven Engineering vs Code-centric (2013)
10. Morgenthaler, J.D., et al.: Searching for build debt: experiences managing technical debt at Google. In: Proceedings of the 2012 3rd International Workshop on Managing Technical Debt, MTD 2012, pp. 1–6 (2012)
11. Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., Meißner, K.: Harnessing the power of semantics-based, aspect-oriented adaptation for AMACONT. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 106–120. Springer, Heidelberg (2009)
12. Nord, R.L., et al.: In search of a metric for managing architectural technical debt. In: Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012, pp. 91–100 (2012)
13. Rossi, G.: Web Engineering: Modelling and Implementing Web Applications. Springer, London (2008)
14. Seaman, C., Guo, Y.: A portfolio approach to technical debt management. In: Proceedings - International Conference on Software Engineering, pp. 31–34 (2011)
15. Svahnberg, M., et al.: A taxonomy of variability realization techniques. Softw. Pract. Exp. **35**, 705–754 (2005)
16. Urbieta, M., et al.: Designing the interface of rich internet applications. In: 2007 Latin American Web Conference (LA-WEB 2007), pp. 144–153. IEEE (2007)
17. Urbieta, M., et al.: Modeling, deploying, and controlling volatile functionalities in web applications. Int. J. Softw. Eng. Knowl. Eng. **22**, 129–155 (2012)
18. Weber, J.H., et al.: Managing technical debt in database schemas of critical software. In: 2014 Sixth International Workshop on Managing Technical Debt, pp. 43–46. IEEE (2014)
19. Whitney, R.: Glorp Tutorial (2005)
20. Zazworka, N., et al.: Comparing four approaches for technical debt identification (2013)
21. Application quality benchmarking repository - Appmarq – CAST. http://www.castsoftware.com/products/appmarq
22. Aspectxml - An Aspect-Oriented XML Weaving Engine (AXLE) - Google Project Hosting. https://code.google.com/p/aspectxml/
23. Drools - JBoss Community. http://drools.jboss.org/
24. FindBugs™ - Find Bugs in Java Programs. http://findbugs.sourceforge.net/
25. HP Static Analysis, Static Application Security Testing, SAST. http://goo.gl/qtRf0X
26. Maven – Welcome to Apache Maven. https://maven.apache.org/
27. phantom @ pleiad.cl
28. Quartz Scheduler | Documentation | Quartz 1.x Tutorials: crontrigger. http://goo.gl/D0tNbO
29. SonarQube™. http://www.sonarqube.org/
30. XSL Transformations (XSLT). http://www.w3.org/TR/xslt